University of London
University College London
Department of Electronic and Electrical Engineering

Doctor of Philosophy Dissertation

Autonomous Grid Scheduling Using Probabilistic Job Runtime Forecasting

Thesis Submitted for the Degree of
Doctor of Philosophy of the University of London

Aleksandar Lazarević

Supervisor: Dr. Miguel Rio

London, 2008

Declaration of Authorship and Originality

I confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Date:

Aleksandar Lazarević

Abstract

Computational Grids are evolving into a global, service-oriented architecture – a universal platform for delivering future computational services to a range of applications of varying complexity and resource requirements. The thesis focuses on developing a new scheduling model for general-purpose, utility clusters based on the concept of user requested job completion deadlines. In such a system, a user would be able to request each job to finish by a certain deadline, and possibly to a certain monetary cost. Implementing deadline scheduling is dependent on the ability to predict the execution time of each queued job, and on an adaptive scheduling algorithm able to use those predictions to maximise deadline adherence. The thesis proposes novel solutions to these two problems and documents their implementation in a largely autonomous and self-managing way.

The starting point of the work is an extensive analysis of a representative Grid workload revealing consistent workflow patterns, usage cycles and correlations between the execution times of jobs and its properties commonly collected by the Grid middleware for accounting purposes. An automated approach is proposed to identify these dependencies and use them to partition the highly variable workload into subsets of more consistent and predictable behaviour. A range of time-series forecasting models, applied in this context for the first time, were used to model the job execution times as a function of their historical behaviour and associated properties. Based on the resulting predictions of job runtimes a novel scheduling algorithm is able to estimate the latest job start time necessary to meet the requested deadline and sort the queue accordingly to minimise the amount of deadline overrun.

The testing of the proposed approach was done using the actual job trace collected from a production Grid facility. The best performing execution time predictor (the auto-regressive moving average method) coupled to workload partitioning based on three simultaneous job properties returned the median absolute percentage error centroid of only 4.75%. This level of prediction accuracy enabled the proposed deadline scheduling method to reduce the average deadline overrun time ten-fold compared to the benchmark batch scheduler.

Overall, the thesis demonstrates that deadline scheduling of computational jobs on the Grid is achievable using statistical forecasting of job execution times based on historical information. The proposed approach is easily implementable, substantially self-managing and better matched to the human workflow making it well suited for implementation in the utility Grids of the future.

To the one who made it all possible

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today, an increasing number of scientific disciplines are faced with problems requiring unprecedented amount of computational power and data storage. Long standing consumers of the CPU cycles, such as high energy physicists and weather forecasters, are now joined by bio-tech entrepreneurs and ground breaking researchers in the arts and humanities fields competing for scarce high-performance computer installations. Equally strong is the need of global commercial enterprises, large corporations and the financial industry for a supply of reliable and resilient computing power coupled to the vast amounts of data storage and high capacity communication links.

The discrepancy between the ability of a single entity to supply the necessary computational resources, and the collective need for tackling the complex problems at hand was the primary motivation for the development of collaborative distributed computing efforts in the last decade. Linking the resources spread out at different academic centres was seen as the best way to capitalise on an investment already made, and as a way of enabling wider access to specialised instruments and valuable scientific data. This concept became known as Grid computing. But the monetary and strategic value of those resources meant that inclusion in the federated pool was acceptable to their owners only if they can maintain a high level of control over their usage and availability.

The loosely coupled distributed environments emerging from these collaborative efforts were, and still remain, hard to manage and support. Crossing administrative boundaries, connecting heterogeneous hardware and using a plethora of technologies, these distributed systems generate an administrative burden severely limiting their adoption. The legacy management approaches inherited from centralised, or rigidly distributed, computing clusters are not suitable for the new dynamic federations of independent resources. As a result, a clear need for an autonomous and intelligent resource management platform has emerged.

1.1 Motivation

In a distributed computing system the scheduling system is the core resource management component responsible for the prioritisation of submitted jobs and their assignment to the available execution nodes. The scheduling principles in the current Grid installations are predominantly based on legacy batch approaches queuing jobs on a first-come-first-served principle, possibly requiring users to explicitly state the maximum allowed execution time of each job. The end effect is under-utilisation due to idle periods, or lower than expected quality of service experienced by the users whose jobs fail to capture the required share of resources. These methods are rigid and poorly suited to a dynamic, service-oriented platform such as the Grid.

The work in this thesis is motivated by a need for a more effective and flexible scheduling system, one that is more closely matched to the users' workflow and able to deliver better exploitation of the future Grid services. The author's view, and the key proposition of this work, is that such added value can be achieved through the use of a deadline and economy based scheduling approach – enabling the user to specify the completion deadline and the available "budget" for the execution of submitted computational jobs. These metrics are embedded in the way services are commissioned in the real world in which users require them to be delivered in certain time and at a defined cost.

From the end-user perspective, this novel scheduling method would enable more flexible working practices and the ability to specify the relative urgency of each job in terms of the deadline "tightness". From the perspective of commercial Grid operators, deadline scheduling could increase the utilisation of their resources, and therefore their return on the investment made, by balancing the peak and off-peak demand. A Grid market could also be supported by the ability to package computational power as a service of a certain quality and deadline adherence levels.

1.2 Objective

Scheduling jobs to a user requested deadline is dependent on the ability to predict the execution time of each queued job, and on an adaptive scheduling policy able to use those predictions to maximise deadline adherence. The objective of the work presented in this thesis was to deliver these abilities in an autonomous and self-managing way, with the least possible impact on the users' workflow and the lowest administrative burden.

1.3 Inspiration

The main role of any scheduling system, computing or otherwise, is in balancing conflicting requirements of consumers and providers of the contended resource being managed. The scheduling process must therefore satisfy resource owners while providing users with sufficiently high quality of service for them to continue using the resource. Proposed probabilistic deadline scheduling was inspired by existing concepts from service-orientated industries, applying them in novel ways to deliver the balance between suppliers and consumers in the context of distributed job scheduling.

Many examples exist of users' willingness to accept services with fuzzy, probabilistic guarantees - whether this is explicitly stated to the user or simply implied in the service offering. Plain old telephone system (POTS) is a prominent example, with low but measurable possibility of call blocking. Chargeable resident parking schemes often used in big cities are an all too familiar example of an oversubscribed resource for which availability is only probabilistically guaranteed. Even in the world of business transactions, commonly associated with very well defined contracts, goods with probabilistic properties can be traded. For example random length timber contracts [1], which are standardised shipments of lumber pieces of various lengths, are listed on the Chicago Mercantile Exchange. The buyer does not know the exact number or length of timber pieces but is buying a shipment which, within some agreed bounds, fits a predefined distribution of lengths. These, and many other examples, show that users are not averse to paying for a probabilistic service as long as it is properly defined and deemed of acceptable value according to the consumers' own judgement.

The concept of deadlines is also well established in the human workflow, and is often the basis of service industry pricing models. The price of many common services, from photo development labs to dry cleaners, is affected by the requested turnaround times. Such pricing structures enable users with flexible deadlines to reduce their costs, while the service providers benefit from a more balanced workload and more efficient use of resources.

Assuming an economically driven view, the service suppliers are predominantly interested in maximising their profit through increased utilisation of their resources. Generally, some degree of over-selling, under-provisioning, or statistical multiplexing is used to boost utilisation past the point possible with hard partitioning and reservations. Since the mid-1990s, the optimisation of resource usage has taken a more pro-active approach through yield management [2] approaches. This concept, greatly facilitated by the use of computers and the Internet, is based on analysing, understating and anticipating consumers' behaviour in order to maximise profits through price or service level differentiation. Yield management was popularised by the airline industry as they manage access to an expensive and contended service whose use should be maximised. Since this is

very similar to a computational utility business model, both of which have an inelastic installed capacity and a seasonal, bursty demand [3], the yield management could similarly be applied to an economy driven utility computing environment*.

The effectiveness of such revenue optimisation approach highly depends on the ability to predict user demand for the services or resources. In that respect, studying past behaviour of consumers has been very effective in obtaining reliable predictions and usable models of their future demand. Among many examples is the use of "loyalty cards" by most large retailers. In return for very detailed statistics of their shopping habits, clients are rewarded with discount points. Although a similar approach could be used to manage the demand for a computational resource, no such effort has yet been made. While the usage statistics of compute clusters are collected and analysed, this is mostly done off-line and in a way that does not sufficiently capitalise on the potential to use this information as a control element of the resource management and job scheduling process.

1.4 Thesis Outline

1.4.1 Contributions

The author's research efforts were concentrated in three main aspects of the work: the analysis of Grid workload, development of a job execution time prediction method and the research into a suitable deadline scheduling algorithm. Correspondingly, the major contributions in these fields can be summarised as follows:

- An extensive characterisation of a year long, multi-purpose, production Grid workload documenting a number of job properties with long-tail behaviour, scale invariance and long range dependency – factors which significantly alter the way such data can be modelled and analysed, consequently invalidating some of the assumptions previously made by other researchers.
- An automated algorithm for identifying job properties available at the time of job submission that can be used to partition the highly variable workload into subsets of "similar" behaviour, thus reducing the variance of job execution times and increasing their predictability.
- A study of the long-term changes of the Grid workload properties through the locality of sampling analysis, and the resulting integration of the job temporal properties into the workload partitioning and the job execution time forecasting work.
- A prediction system, using automatically parametrised time-series forecasting methods, to estimate the execution time of queued jobs based on their historical performance and associated job properties.

*A research proposal studying the application of yield management in compute utilities based on the methods documented in this thesis has been submitted to BT, see Appendix C.7

- A novel Grid scheduling approach, previously applied in the context of real-time systems, which uses the estimates of job runtimes to calculate the latest start time necessary to meet the requested completion deadline.
- A study of the commercialisation potential of predictive, probabilistic and deadline based Grid scheduling as applied to commercial utility computing service providers analysing the Grid value chain, possible exploitation routes and offering an in-depth argument for developing a scheduler add-on component.

1.4.2 Publications

The research contributions in this thesis led to the following publications:

1. A. Lazarević and L. Sacks, **"Managing Uncertainty - A Case for Probabilistic Grid Scheduling"**, *Proceedings of The Seventh International Meeting on High Performance for Computational Science - VECPAR 2006*, Rio de Janeiro, Brazil, July 2006.
2. A. Lazarević, L. Sacks and O. Prnjat, **"Enabling Adaptive Grid Scheduling and Resource Management"**, *Proceedings of The Ninth IFIP/IEEE International Symposium on Integrated Network Management - IM2005 - Application Session*, Nice, France, May 2005.
3. A. Lazarević and L. Sacks, **"A Study of Grid Applications: Scheduling Perspective"**, *Proceedings of The 2005 London Communications Symposium*, London, UK, September 2005.
4. A. Lazarević and L. Sacks, **"Lightweight Scheduling for Grid Applications"**, *Next Generation Networking: Multi-Service Networks Workshop*, Abingdon, Oxfordshire, UK, July 2005.
5. A. Lazarević and L. Sacks, **"Measuring and Monitoring Grid Resource Utilisation"**, *Proceedings of The 2004 London Communications Symposium*, London, UK, September 2004.
6. A. Lazarević and L. Sacks, **"Adaptive Grid Scheduling and Resource Management"**, *Next Generation Networking: Multi-Service Networks Workshop*, Abingdon, Oxfordshire, UK, July 2004.
7. I. Liabotis, O. Prnjat, T. Olukemi, A. Lazarević, A.L.M. Ching, L. Sacks, M. Fisher and P. McKee, **"Self-Organising Management of Grid Resources"**, *Proceedings of The International Conference on Telecommunications - IST2003*, Isfahan, Iran, August 2003.

8. A. Lazarević and L. Sacks, **"Resource and Application Models for Advanced Grid Schedulers"**, *Proceedings of The 2003 London Communications Symposium*, London, UK, September 2003.

1.5 Thesis Organisation

This introductory chapter laid out the primary motivation for the thesis, defined its objective and offered some real-world inspiration for the proposed approach. An outline of the primary research contributions and the resulting publications were also given. The rest of the thesis is structured as follows.

Chapter 2 offers a general background to distributed computing and the Grid. This chapter also introduces the overall, high-level, methodology of the work and presents the thesis' scope, limitations and assumptions made. The work also briefly discusses in the context of the sponsoring research projected.

Chapter 3 gives an overview of the previous research work in the fields of cluster and Grid scheduling, workload characterisation and performance predictions. By defining the problem space for each of these topics, and by outlining previously proposed solutions and their implementations, the chapter points to the inability of the current scheduling implementations to successfully fulfil users' expectations, and to the pitfalls of current methods for predicting job execution times. The workload characterisation section will survey previous work on the topic, which was based on older, pre-Grid job traces, and will serve as a comparison to the properties of the Grid workload analysed later in the thesis. This chapter concludes with the survey of past work on Grid monitoring and simulation tools, two important aspects of Grid usage data acquisition and scheduler testing.

Chapter 4 presents the findings of the characterisation study done on a 12 month workload trace collected from a multi-purpose production Grid facility at University College London. Motivated by the need to better understand the behaviour of the workload and its long-term evolution, the study looks not only at the common analysed metrics (such as the arrival process, queue wait times etc.) but also at the correlation of different job properties and their execution times. By investigating those functional dependencies, the study indicates the candidate properties for job partitioning that would lead to a reduction in data variability and an increase in job execution time predictability. The analysis also considers changes of job properties through time and their variation caused by differently sized sampling windows as presence of any such temporal locality is an important factor in the selection of the appropriate forecasting model.

Chapter 5 considers the prediction of the length of job execution based on the job properties available at the time of submission and the historical model for "similar" jobs. An automated method for job partitioning based on the exhaustive search for the combination of job properties leading to the greatest

reduction in the coefficient of variation is proposed. The chapter presents a comparison of five time-series based, and automatically parametrised, predictors and discusses their forecasting accuracy by using appropriate error metrics of different robustness and sensitivity.

Chapter 6 introduces a novel deadline scheduling algorithm for computational Grids based on the earliest deadline first method previously used in the context of real-time systems. The performance of the scheduler is evaluated through a simulation using the trace of actual Grid jobs, two deadline generation methods and two job execution time predictors. The chapter demonstrates that effective deadline scheduling is achievable using the proposed scheduling algorithm and job execution time estimation methods.

Chapter 7 discusses specific previous work most closely related to the approaches presented in this thesis. It offers important distinguishing aspects between them and compares the findings and results obtained. This chapter also motivates the discussion on the outstanding issues related to the thesis work and the direction of further improvements which are given in Chapter 8. Finally, the thesis concludes with the summary of findings in Chapter 9.

The thesis contains several appendices offering additional support to the arguments put forward, or providing further information on the work undertaken. The author's contributions to the sponsoring research project are summarised in Appendix A. The effects and behaviours observed in characterising the Grid workload are further supported through the analysis of an additional Grid usage trace presented in Appendix B. Appendix C, sponsored by the London Business School and the Centre for Scientific Excellence, examines the business potential of this research thesis and proposes a possible commercialisation route.

Chapter 2

Background

This chapter opens with an introduction to distributed computing and the Grid, followed by the high level research and implementation methodology, the definition of the scope of the thesis and an explanation of the assumptions made and limitations set. The chapter concludes by placing the work in the context of the EPSRC* funded research project to which the author has contributed.

2.1 Distributed Computing and the Grid

Despite being actively considered since the 1980's, distributed computing is still a very dynamic field of research and development. Grid computing, the latest distributed platform, offers exciting new opportunities, but some unique challenges as well.

2.1.1 Historical Perspective of Distributed Computing

In 1997, advances in computer networking technologies led the Legion Project [4] team to propose a model for unifying geographically distributed compute resources into a common platform. Several similar ideas were considered in the research community for years, and have been sporadically used in the academic circles, but the first project to popularise wide area distributed computing was the screen-saver based search for extraterrestrial intelligence running on idle Internet connected PCs (SETI@Home [5] started in 1999).

Today, distributed computing is increasingly being used not only for its performance benefits, but also due to good scalability and resilience it can provide. Legacy high performance distributed installations used specialised parallel processing hardware and proprietary low latency networks to run highly optimised applications. While such systems do still serve a specific niche, the majority of

*Engineering and Physics Sciences Research Council

the contemporary compute workload is now done by the increasing number of high throughput clusters, made using widely available components, connected via ubiquitous IP networking [6]. From web servers to financial risk analysis, these distributed systems are often based on the open-source software and use either a "cycle scavenging" method (such as Condor, see Section 3.1.3), or some implementation of the distributed master-worker middleware (like the Sun Grid Engine, Section 3.1.3).

Recent interest in distributed computing is being driven by both commercial and educational sectors. In the academic institutions, a shift into extremely computationally demanding "Big Science" [7] requires investment in infrastructure often beyond reach of even the most developed nations, thus fuelling cross-border collaboration efforts. Businesses are eager to deploy distributed solutions that will enable them to better use their installed capacity and increase resilience and agility by unifying their compute platforms. The distributed computing approach, while having potential to fulfil most of these requirements in the long term, has often been a victim of its own success, oversold by its enthusiast and hampered by the lack of adequate enabling technologies[8].

2.1.2 Grid Computing

With the proliferation of high bandwidth networks, their almost universal interoperability, the reduction in the cost of data storage and an increased portability of applications between the platforms, the technological gap inhibiting truly globally distributed computers was being closed.

By using these enabling technologies, Grid computing [9, 10] was based on a primary objective to develop a transparent and portable middleware able to integrate heterogeneous resources into a distributed computing platform. The Grid was developed as a much more dynamic environment than its predecessors, able to form transient, on-demand Virtual Organisations (VO) [9] spawning geographical, networking and administrative boundaries. This middleware would link distributed computational, storage and visualisation resources into persistent environments, provide a strong security layer, and a standardised methods for discovering available resources and their capabilities.

The novelty of the computational Grids was in their aim to offer compute power as a utility, a service to the consumer paid on a per use basis. In this aspect they drew significant inspiration from the electricity power grids, trying to decouple resource generation from the transmission network. Migration to the service orientated approach would have some important implications for the end user. Compute capacity would be available as and when required, reducing the need to dimension local resources for peak usage and thus lowering capital expenditure. Users would be more agile and able to react more quickly to the changing computational priorities. The standardisation would lead to a development of a

Grid services market, boosting the competition and producing economies of scale that drive the reduction in cost. But these benefits would come at the expense of relinquishing direct control over the hardware and software, fully relying on the security provided by the middleware and the service functionality offered by the supplier. It is then no wonder that primary obstacles in embracing the Grid are not technological but social [8].

A future computer usage scenario, supported by distributed computing services, would see a broad mix of consumers, from the casual users to the large institutional entities each with its own computing requirements and Quality of Service expectations, connect through a broadband network to a computing platform on which they could execute their compute jobs. The cost of the service would be dictated by the supply and demand in the Grid market economy, and the price influenced by the requested level of service, urgency of the job, its complexity, and other factors. However, a number of open issues and problems would have to be solved before such transparent use of the compute resources becomes feasible.

2.1.3 Open Issues and Problems

Extensive research of distributed computing approaches undertaken in the 1980s and 90s has yielded proven solutions for many of its implementation and programming problems. Despite the similarities and common roots to the legacy distributed computing, the Grid poses radical new challenges and requires novel approaches for solving them. The primary added value of the Grid, its ability to supply computing power as an on-demand service through a semi-persistent environment created for solving a specific task (VO), is in stark contrast with the legacy cluster systems and their strict "plan-deploy-use" cycle. Therefore, legacy approaches and solutions cannot simply be migrated onto the Grid middleware, as they would diminish the core benefit that this new technology has to offer.

The Grid's envisaged flexibility to operate on the time and/or space shared hardware, interconnected by dedicated or contended networks, and across administrative boundaries adds a whole new layer of complexity to its management. It follows that in developing the core Grid middleware components, one should assume little of the operational environment, and require even less, aiming for an adaptable system able to operate in a wide range of conditions.

Resource Management Problem

After the initial research effort to develop and deploy the first Grid services, the problem of managing systems of such global scale became apparent [11]. This large administrative burden is caused by the scale and heterogeneity of the platform, outdated management tools, and the reluctance to radically change management practices. Desirable properties of any new Grid middleware components

would therefore be a high degree of autonomy and self-management, and a low impact on the end users and their workflow.

Grid Workload Properties and Scheduling Process

Future development of the Grid middleware will be greatly influenced by the nature of the applications that run on it. The Grid has already enabled scientific simulations and experiments to be performed at the previously impossible scale, but as it becomes a widely accepted collaborative computing platform the application set is likely to change. With the development of computational markets [12], users could find it cheaper and more convenient to use the Grid for an increasing variety of jobs. The grid may emerge as a generalised service delivery platform with a very diverse application set, executing large numbers of medium and low complexity jobs mixed in with few high demand ones.

Any such changes in the usage profiles would change a number of important job statistics which current management components rely on. As the applications execution times fall, job arrival rates will increase, and so will the resource discovery and scheduling overheads. Current Grid resource discovery and scheduling components are built on assumptions of a very long execution times and the resource pools of modest size. Overheads and job submission delays now introduced by the Grid middleware may be considered insignificant, but in the future may represent the greatest part of the job execution time. In a general use case, schedulers will have to make an intelligent decision and adjust the complexity of the resource discovery and scheduling to the likely complexity of the job at hand.

Resource Monitoring

Scalable monitoring of the Grid is difficult due to its heterogeneous nature and a large number of resources that need to be observed. Monitoring systems with predefined sampling points and frequencies will inevitably end up with poor information capture, high volumes of irrelevant measurements in which a truly important observation, and its cause, may be lost. Operating in a geographically distributed environment, transferring monitoring information indiscriminately leads to an inefficient use of bandwidth. The next generation of truly effective Grid monitoring systems would have to be more intelligent, flexible and agile, adapting the granularity, frequency and the communication methods to the state of the operating environment and the importance of the measurements. These systems would not be unlike virtual sensor networks, permeating the Grid fabric and self-organising in monitoring constellations according to the current requirements.

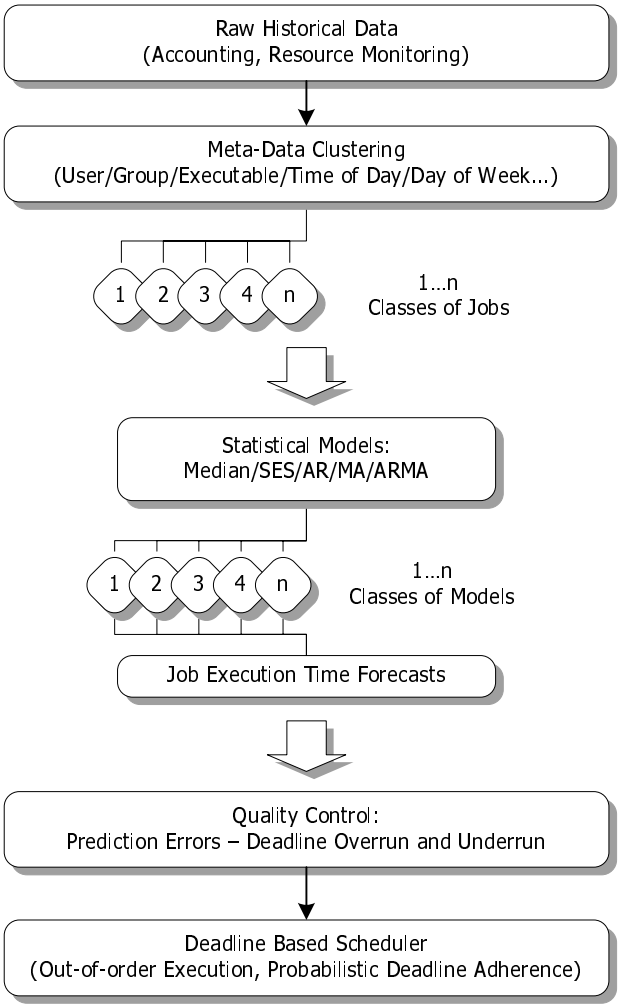


Figure 2.1: Overall Methodology Diagram

2.2 General Research and Implementation Methodology

The overall proposed methodology for delivering deadline scheduling is shown in Figure 2.1. The basis is the on-line use of the historical job resource usage data collected by the monitoring and accounting elements of the Grid middleware. This data is analysed and mined for patterns, correlations and functional dependencies between the past job execution times and the job properties (also referred to as the job meta-data) which are available to the scheduler at the time of the job submission or while the job queues for resources. These properties include, but are not limited to, the identity of the user submitting the job, the Virtual Organisation to which the job belongs, the name of the job executable and its parameters, the time of the day or day of the week of job submission etc.

A workload analysis and similarity-based partitioning method, developed as part of this thesis, identifies a combination of one or more job properties that are used to separate the workload into a number of classes with a lower variability of job execution times than the entire workload had. An example could be a

class of jobs owned by one of three different users, with a given executable name and mostly run on workday afternoons. A statistical model of a significantly better fit and a much higher accuracy can then be used to forecast the future execution times of jobs in that workload class than it would be possible without such similarity grouping.

Job execution time predictions are the essential enabling element of the deadline scheduler implementation. By anticipating the execution time of the queued jobs, the scheduler is able to calculate the latest job start time for a certain user requested deadline, and can use this information to dynamically prioritise jobs with "tighter" deadlines. The forecasting performance, and the deadline overrun and underrun statistics, could be fed back to the prediction model and can be used to increase its accuracy, or change the way the workload is partitioned in response to a significant shift in the usage patterns.

The following sections will discuss the high level methodology of the three main aspects of the work: workload characterisation, job execution time forecasting and deadline scheduling. A more detailed discussion of the specific methodology, implementations and approaches used for each of these three main areas is offered in the separate sections in Chapters 4, 5 and 6.

2.2.1 Workload Characterisation

The essential first step in the pursuit of good job runtime predictions was to thoroughly analyse and understand the properties and specific features of the data set that will be forecasted. Parallel and distributed workload characterisation was the subject of significant amount of previous research (which is surveyed in Section 3.3), but was mostly based on a limited number of workload traces collected in the 1990s and made available through the Parallel Workload Archive*. With the emergence of the Grid, distributed computing has taken a more dynamic form, adding some new features strongly differencing it from the traditional parallel clusters. These differences, that will be discussed in more detail in Section 3.1.1, meant that a new and more representative workload should be used to judge the changes that this new approach, user base and workflow have introduced.

In 2003 the Grid technology was just emerging from the research facilities and into the production use, Grid installations were few and limited to the testbeds and single, specific and limited use facilities. Several large projects were federating these Grids into larger communities, and the decision to install a Grid cluster at the University College London opened the possibility of obtaining relevant and representative usage data from one of the first Grid connected clusters used by a number of different research projects from within UK and abroad.

Considering previous workload characterisation studies from the aspect of job execution time predictions, it was evident that the variability of the data set was

*Available at http://www.cs.huji.ac.il/labs/parallel/workload/

very high and that modelling this whole dynamic range would lead to very poor results. Methods for reducing the variability of the data were required and had to be based on the information available to the scheduler at the time the job was submitted. For this purpose, the wealth of the meta-data collected by the Grid accounting and monitoring systems was used to look for links between the job execution times and the job's name, its properties, the submitting identity, and for the first time, its temporal characteristics such as the day and the time the job was submitted.

Correlation between these parameters was anticipated due to the nature of the human work cycle which is the major contribution to the system workload. While the Grid as a whole may be geographically distributed, individual users reside in a certain geographical area and will have a daily and weekly work cycle specific to their location. They will also more likely work on one or two scientific projects at the time and tend to run applications relevant to those efforts. They may also have some specific workflow habits, and with their own intuition (or expectation) for the length of the execution of their jobs, they might be submitting more complex jobs to run overnight or during their lunch break. While these effects may not be visible when looking at the aggregate load generated by a large number of users, partitioning the data according to one or more of these criteria would likely reveal the distinct usage patterns. Understanding these features would prove instrumental in devising a suitable forecasting methodology.

Contrary to the characterisation studies whose aim was to capture the properties of the trace in a model suitable for generation of other, different but statistically representative models, the aim of the workload characterisation presented in this thesis was to establish the models suitable for the ongoing prediction of the job execution times. Such an approach cannot simply treat the workload as a snapshot in time, but requires the analysis of its dynamic properties and its changes through time. Therefore, Section 4.6 looks at both the low frequency, gradual evolution, and the high frequency sudden and abrupt changes in the job properties. The gradual changes are more characteristic of an ongoing development of the workload, such as a growing scientific data set being analysed, for example. The more abrupt discontinuities are indicative of a change in the application, data set or the simulation goal, or perhaps a transient hardware or software failure. All such events are intrinsic parts of a real world system, and while they may justifiably be excluded from a generative model, they must be considered in the creation of a robust prediction approach.

Engineering this robustness into the system and testing it under realistic conditions depends on knowing what to expect in terms of the statistics and distributions of the job parameter values. Some important previous work on the predictive scheduling, discussed in more detail in Section 7.2, has used simple approximations of the critical job properties which may not reflect the reality of the Grid computing. The distribution functions, and their properties, of all

the relevant job parameters were examined and special attention was paid to the presence of long tails*[13] or self-similarity† [14]. The presence of such statistical features invalidates some of the previous approaches which did not take them into account, while at the same time influencing the design of future, robust scheduling systems.

2.2.2 Job Execution Time Predictability

With the benefit of having access to a multi-purpose production Grid, and the ability to collect usage data on this facility, the use of this real world workload was favoured over the synthetic traces generated using one of the several workload models and generative algorithms. Therefore, the aim of the job execution time predictability study was to assess the accuracy level to which this actual, production Grid workload could be forecasted.

The analysis and characterisation of the Grid usage data, and especially of the workload partitions generated using the identified pivot job properties, indicated different statistical properties of the job execution times between these job groups. Most importantly, while the largest number of job partitions exhibited strong autocorrelation properties, some execution times were resembling a random and mean-reverting process. The use of a single forecasting method was therefore not advisable, and several time-series and mean based predictors were considered.

A further reason for using multiple prediction algorithms was that in the online forecasting, the prediction speed could be as important as the prediction accuracy. In the case of probabilistic scheduling some short jobs may be assigned a model of lower complexity and accuracy, while the longer running jobs may warrant a highly complex but accurate model to reduce the effect of the prediction errors.

The time series methods selected for predicting job execution times include simple exponential smoothing (SES), auto-regressive (AR), moving average (MA) and the auto-regressive moving-average (ARMA) methods. Sliding window median was included to predict the non-autocorrelated series and was favoured over the mean predictor due to its robustness against outlier values. All of these methods will be fully described in Section 5.2.2.

The important aspect in implementing all of these forecasting algorithms was the level of self-management, adaptation and robustness that can be built in. The system was envisaged as an autonomous entity requiring the minimum of administrative attention and no input from the user (apart from the desired deadline). This motivation led to the development of an automated process of

*A colloquial name for a feature of some statistical distributions in which the high-frequency population is followed by a low-frequency one that gradually "tails off" but can still make up the majority of the area under the probability density curve.

†An object or a process which is exactly or approximately similar to a part of itself.

workload analysis, selection of the job properties used for workload partitioning, and model parametrisation further discussed in Section 5.2.

In dealing with robustness, the thesis also takes a somewhat holistic view that no observed feature of the workload should be considered as an anomaly or exception. Rather than removing these "misbehaving" jobs, as recently suggested by some workload characterisation studies [15], the choice was made to attempt to proof the system against such departures from the modelled behaviour. In real life, hardware and software does crash and user behaviour can at times seem erratic. While a generative workload model can afford to ignore such events, a predictive one has to deal with them in the best possible way.

Finally, the quality and the accuracy of the forecasts should be judged with the appropriate statistical measures that enable adequate comparison with other work in the field. Unfortunately, much of the previous job execution time prediction work selected these metrics based on habits and personal preferences, rather than on the statistical properties of the forecasted series or the measured prediction accuracy. In this thesis, measures of different sensitivity, robustness and scale dependence were employed and their use was thoroughly justified in Section 5.2.3.

2.2.3 Deadline Scheduling Methods

Once estimates of the execution times of queued jobs are known, suitable scheduling algorithms can be used to order them in such a way as to maximise the adherence to the requested deadlines, minimise the overrun time or optimise the profitability of the cluster for the Grid operator. The aim of the thesis was not to develop a software component for any specific Grid middleware, nor was it to engage in an in depth assessment of the deadline scheduling policies. The focus was on establishing a proof-of-concept "prediction engine" that could be interfaced to an existing scheduler which is able to make use of this information. Several such schedulers (discussed in Section 3.1.3) make provisions for the job execution time forecasts but either do not generate them internally or do so in a trivial manner.

The thesis does propose a scheduling algorithm not previously used in the context of Grid computing, and in Chapter 6 establishes its performance through a trace-replay simulation using actual production workload. The results obtained serve as a justification of the efforts to predict the job execution times, as well as a motivation for further work on the development of better and more efficient implementations of the Grid scheduling policies.

2.3 Thesis Scope, Assumptions and Limitations

The focus of the work presented in this thesis is maintained by a well defined scope of both the platform and the service to which the proposed job execution time forecasting approach will apply. The work also makes some assumptions to the way the Grid installations will be deployed and the Grid services used. This section will present the scope, and those assumptions, together with some necessary limitations to the considered research area.

2.3.1 The Platform

The primary motivation of the thesis, the adopted high level approach and the stated methodology are universally applicable to distributed cluster computing. However, some specific challenges, functionality issues and implementation problems are considered in the context of delivering deadline scheduling on the Grid, the latest and most commonly accepted wide area distributed computing platform [16].

The Grid platform assumed in this thesis is not seen as a highly specialised, custom built state of the art facility, but rather as a metaphor for a broader general purpose utility computing installation. These Grids are commonly built using commercial-of-the-shelf (COTS) components and standardised architectures to minimise their procurement costs. The focus in these systems is on the ease of the life-cycle management, as the reduction in the cost of the hardware is often reflected in the increased system administration and maintenance expense.

Clusters federated into the Grid environments are often heterogeneous, and the ability of the Grid middleware to integrate these disparate entities into a coherent platform was one of the primary driving factors for its adoption. But within the clusters, and especially commercial and production ones, every attempt is made to keep the hardware homogeneous due to the easier resource management and significant savings that can be made through economies of scale.

2.3.2 The Service

The assumption of the thesis is that a future commercial Grid utility operator, such as the recently started Sun Utility Compute* or Amazon Elastic Compute Cloud, would serve geographically distributed users from administratively and functionally diverse communities. These consumers of compute power would execute a mix of everyday personal and networking software, as well as some intensive business and scientific workload. The users would thus require computational power for a full range of applications from low complexity repetitive tasks to highly demanding specialised workflows.

*http://sun.com/grid/

This on-demand utility computing service would have a very dynamic usage profile consisting of both continuous streams of jobs and bursts of activity. In this environment, the quality of service and the contractual obligations would be governed by service level agreements (SLA). These contracts already give a probabilistic guarantee of the service availability (such as 99.9% uptime) or the delivered performance level (average packet delay of 20ms for example), and could be easily extended to include a probabilistic deadline adherence guarantee as well (for example at least 95% of made deadlines and average deadline miss time of 1000 seconds).

2.3.3 Limitations

In considering the job execution times, the influence of the past or future network performance is not directly taken into account. This aspect has, beyond doubt, strong influence on the runtimes of jobs dependant on the network for data transfers, synchronisation or interaction with the user. However, modeling of the local and wide area network performance, and its influence on the jobs running on distributed platforms, was subject of extensive previous research [17, 18, 19, 20]. Most prominently, the Network Weather Service [21] was uses by Wolski to judge the execution time of jobs under different network conditions in [22].

The execution time forecasting algorithm proposed in this thesis does have some sensitivity to the varying network performance through the influence this has on an I/O bound job. If the runtime of such a job is predominantly influenced by the network performance, which was previously shown to be correlated with the daily and weekly work cycles, the resulting model will exhibit the same behaviour and in effect predict the performance of the application as the function of the network performance. Further work could also consider incorporating the network performance metric as another job property taken into the consideration alongside other meta-data.

Another performance influencing element that has been extensively researched and that was not considered in this thesis is the influence that the number of assigned processors, often referred to as the size or the degree of parallelism of a job, has on its execution time. The dependence between the size of the job and its runtime was previously modelled by Cirne and Berman [23] and others [24, 25, 26, 27, 28].

However, as far as it is possible to tell from the available data, the Grid workload at the cluster level tends to be composed of single CPU “bag of tasks” jobs. Apart from the trace collected by the author, the only other publicly available workload from the largest European production Grid (the EGEE project [29]), contains a quarter million jobs from a ten month period all of which requested a single processor. The lack of the multi-processor jobs visible

on the cluster scheduling level certainly does not mean no parallel jobs are run on the Grid. It rather implies that the complex workflow of parallel and inter-dependant jobs is handled by a higher level meta-scheduler* which plans, partitions and deploys the tasks onto the available resources. Job execution times forecasted by the probabilistic scheduler presented in this thesis could also help the meta-scheduler make more efficient decisions.

As previously justified, this work assumes a relatively homogeneous hardware environment, and hence a balanced performance from all of the worker nodes within the cluster. It is also presumed that the hardware is not time-shared with users external to the Grid, or if it is, that this is under the control of some local low level job scheduler. This may not be representative of some cycle-scavenging Grid middleware (see Condor in Section 3.1.3), but is a reasonable assumption in the view of this work's primary target platform.

Finally, the whole deadline scheduling approach relies on the user supplying "reasonable" deadlines, and being motivated to extend these deadline as far into the future as they possibly can. Without such motivation, users could simply request all jobs to complete immediately which would reduce the deadline scheduling system into a batch first-come-first-served one. The diversity of deadlines can most reasonably be effected through a charging system which would impose higher prices on shorter deadlines and peak usage times. These Grid economy systems have been suggested for some time by Buyya [30, 31, 32, 33], Ernemann [34] and others [35, 33], and fall outside the scope of this thesis. However, the pricing policy of the Grid resources requires an in-depth knowledge of the ways these are used, and the extensive workload characterisation given in Chapter 4 will provide a valuable input.

2.4 Project Context: Self-Organising Grid Resource Management

Research work presented in this thesis was done under the auspices of the EPSRC funded Self-Organising Grid Resource Management (SO-GRM) project, and in collaboration with BT Research (formerly BTExacT). SO-GRM is a base research project aimed at developing an autonomous management infrastructure able to support Grid job execution through its full life-cycle: from job admission through scheduling and resource discovery to security monitoring. Components of the SO-GRM architecture share the same objectives of removing single points of failure through a distributed approach, reducing the administration load by using policy

*Grid scheduling hierarchy is further discussed in Section 3.1.2.

based management and creating an agile, on-demand system through the use of self-organising principles.

The SOGRM management architecture [36, 37] is based on a light-weight, adaptive, and policy-controlled XML-enabled management elements. These are seen as an add-on to the established Grid platforms such as the Globus [38, 39], but are equally applicable and easily integrated into other Grid middleware. Project work has focused on the three issues of primary concern in the Grid management: resource discovery, security and intrusion detection and predictive scheduling, the topic of this thesis. The author's contributions to the project are outlined in Appendix A.

Self-Organising Resource Discovery (SORD) [40] is tasked with the discovery of computational resources which satisfy the conditions set forth by the SLA management component and the scheduler. SORD is a query-response distributed protocol based on the node communication links in a small-world topology [41]. These topologies have previously been considered in the problem of routing with local information and allow distribution of the information to the correct recipient through the use of network shortcuts. The protocol's main design objectives were scalability and resilience to single node failures, both of which have been successfully met. More information on the scalability and the resource discovery success rates can be found in the previous publications by Liabotis [37, 40].

Integrity Information Intelligence (I^3) [42] is a distributed run-time intrusion detection system that combines the anomaly and misuse detection components. After initial training with the features of a well behaving process, the I^3 is subsequently able to recognise suspicious CPU utilisation patterns. The feature set defining an anomaly is stored locally, with all other nodes in the network immunised by broadcasting the anomaly's definition as an XML antidote. In both simulation and testbed deployment the I^3 has provided process classification with less than 1% error rate for a suitably configured threshold detection value. More information can be found in [43, 42].

Chapter 3

The Grid and Related Technologies

This chapter examines the previous work, published literature and the background research done on the topic of (Grid) scheduling and the related technologies. It adopts a top-down approach by firstly treating the issue of job scheduling before examining the past research done on predicting the resource performance and job runtime. The chapter finishes with a systematisation of the workload characterisation studies previously undertaken, and an overview of Grid monitoring tools and simulation suites currently being used.

3.1 Cluster and Grid Schedulers

A scheduler is one of the primary elements of a resource management framework of any computational system. It controls the order in which requests for a contended resource are processed, while ensuring certain performance, reliability or security criteria are met. Packet scheduling on the communication links and task scheduling on the processing units are some of the common examples.

Scheduling is usually performed on several levels, each being more granular and having a tighter control of the resources than the previous one. In a distributed computing system, users submit complex jobs consisting of many, possibly interdependent, tasks which are to be scheduled on the remote clusters. Local job managers schedule those tasks onto the worker nodes within the cluster, possibly together with the locally submitted jobs, and each node does further scheduling of the system and user processes on the kernel level. When the distributed systems consist of heterogeneous, non-dedicated hardware with dynamic availability, and are connected via variable speed, congested links, the scheduling problem becomes very complex.

This section will open with some formal definitions of the scheduling problem and its complexity, followed by a taxonomy of the current Grid scheduling algorithms and a discussion of some of the challenges of scheduling in the Grid context. The implementations of the Grid schedulers, and their strengths and weaknesses, will be presented before concluding with the current state of the Grid scheduling research and a summary of the open issues.

3.1.1 Grid Scheduling Problem

Scheduling in the Grid context is a process of mapping a set of submitted jobs to the available resources, in such a way as to maximise a certain scheduling benefit function, for example the job makespan*, cluster utilisation or similar.

Scheduling Process and Components

Despite a Grid being a platform of high diversity, both in terms of the hardware and the applications, a common high level logical architecture of scheduling components can be constructed, Figure 3.1.

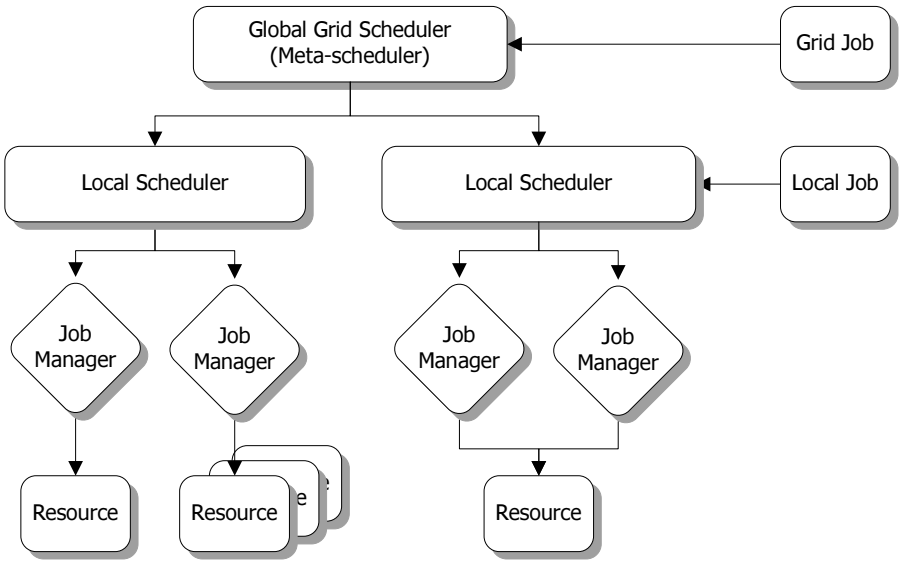


Figure 3.1: A high level diagram of the Grid scheduling components and their interaction.

In this scheduling hierarchy, a Grid scheduler (or sometimes referred to as a meta-scheduler) accepts incoming jobs from the authenticated Grid users, selects a subset of nodes matching certain application requirements from the resource pool advertised, and generates a task-to-resource mapping which is passed to the launching module (or job manager). Contrary to the schedulers in traditional distributed systems, the Grid schedulers do not exercise total control over the

*Time taken from the job submission to the job completion, usually equals queue wait time plus the job wallclock execution time

Grid resources which are often in different administrative domains. The Grid schedulers must work like agents or brokers, with non-exclusive access to these shared resources, and subject to a range of local security and resource utilisation policies. Although a Grid level scheduler is not strictly required, there is little doubt such high level component is needed to successfully harness the potential of the large number of distributed resources. The following discussion assumes that at least one such meta-scheduler is used.

Grid schedulers communicate with a local resource manager in charge of each Grid node using a common protocol (such as Globus GRAM [44] for example). The responsibility of the local managers is to handle the job scheduling from the Grid and the local users alike, and to report the job status, resource utilisation and other accounting data back to the Grid level scheduler. These local resource managers are controlled by the resource owners and the Grid schedulers have no influence over their operation, job prioritisation or the scheduling policies. An overview of the Grid schedulers and job managers is given in Section 3.1.3.
The scheduling process can also be generalised into the following three stages:

- *Resource Discovery* acquires a list of the available resources and their static and dynamic properties such as the CPU clock frequency, operating system or the current memory usage. This is usually done through a Grid information system, of which Globus Monitoring and Discovery System (MDS, [45]) is an example. Alternatives have been proposed [46, 47, 48, 49], including one from our own research group [37].

- *Schedule Generation* maps applications to the resources maximising a certain benefit function. This is the core of the scheduling process and will be discussed in more detail in the following sections.

- *Job Staging and Launching* executes the job mapping supplied by the scheduler by staging the necessary data onto the target resource, submitting the job to the local resource manager using a compatible protocol, and monitoring the job execution throughout its life cycle. The Globus Resource Allocation Manager (GRAM [44]) is the most often used protocol with a number of proxies for communication to the other local resource managers (such as Condor ClassAds [50]).

Challenges of Grid Computing

The general scheduling problem, with its roots in the control theory and optimisation techniques, has been extensively studied as part of many common problems in technology, computing and engineering. In the context of the parallel and distributed systems, the scheduling algorithms have evolved together with the underlying hardware, from vector and massively parallel processor machines to the clusters of commodity workstations today. Although this work can

serve as a source of inspiration, the traditional scheduling approaches create poor Grid schedulers. This is mainly due to the following assumptions these legacy schedulers make:

- the scheduler has exclusive control of the resources,
- all resources are within a single administrative domain and subject to a single set of policies,
- the resource pool is invariant, bar certain exceptional events such as node crashes,
- the contention caused by the incoming application can be managed, and performance offered by the cluster well predicted,
- data staging time is deterministic.

However, most of these assumptions do not hold in a Grid computing scenario. Specific properties of the computational grids, as discussed below, create additional challenges and require novel methods to deliver effective scheduling.

Heterogeneity of computational, storage and network resources leads to different capabilities, different service levels and different specific scheduling policies required. Similarly, a widely varying collection of users and applications present a heterogeneous load with a variable demand and expectations. A Grid scheduler must be able to deal with this level of heterogeneity in a robust and scalable manner.

Autonomy of resources, resulting from the principle that the owner maintains control of its hardware, leads to a diversity of local resource management and access control policies. As the Grid scheduler can exhibit little control over these, an adaptable approach is needed to ensure a low barrier for connecting the resources into the Grid.

Dynamic performance is manifested through constant fluctuations in the availability and service levels of all the resources connected to the Grid. Generally autonomous and non-dedicated, computational, storage and network resources are contended for by other (local) users of the system. The Grid scheduler must monitor these dynamic properties and, if not anticipate possible problems, at least react to the observed changes.

Data staging is increasingly complex with the separation of the data, applications and the target execution nodes. Interconnected by wide area networks, these three points can have a significant communication cost and overhead in between. The Grid scheduler should be aware of the time and cost required to

join these three components, and use that knowledge when selecting the most appropriate schedule.

3.1.2 Grid Scheduling Algorithms

The scheduling problem, as applied to the parallel and distributed systems, has been treated extensively in the seminal works by El-Rewini [51, 52] and Shirazi [53, 54]. This section will open with a discussion of some of the important aspects of the scheduling problem, such as its complexity, and continue to give a taxonomy of the present scheduling algorithms. It will also outline the current approaches to treating the added complexity of the Grid scheduling in the fields of dynamic resource performance and the scheduling benefit functions. The overview aims to present a balanced and encompassing view of the current state of the art, while focusing on the algorithms and approaches of special interest to the dynamic, performance driven and predictive approaches.

Complexity of the Scheduling Problem

The multiprocessor scheduling problem, as a sub set of the scheduling and sequencing of jobs, is an NP-complete optimisation problem [55]. The problem statement is as given in the following:

> Given a set of J jobs where job j_i has length l_i and a number of processors m, what is the minimum possible time required to schedule all jobs in J on m processors such that none overlap?[56]

The formal definition of the NP-completeness was given by Cook in 1971 [57]. In complexity theory, the NP-complete class of jobs are the most difficult problems in the non-deterministic polynomial time (NP). Potential results of these problems are easy to verify for correctness, but no significantly faster method for solving these problems then to try all the possible results has been found. For non-trivial problems, all known algorithms for solving the NP-complete problems require time that is super-polynomial in the input size.
Therefore, one of the following alternative methods are used to solve NP-complete problems:

- *Approximate:* An algorithm that quickly finds a suboptimal solution within a given range of the optimal one.
- *Probabilistic:* An algorithm that can be proven to yield a good average runtime behavior for a given distribution of the problem instances.
- *Heuristic:* An algorithm that works “reasonably well” on many cases, but for which there is no proof that it is both always fast and always produces a good result.

The taxonomy of scheduling further discusses the use of these methods in parallel and distributed systems scheduling.

Taxonomy of Grid Scheduling Algorithms

Casavant proposes a hierarchical taxonomy in [58] for scheduling algorithms in the general purpose parallel and distributed systems. Treating the Grid as a subset of such systems, the Figure 3.2 presents the current approaches.

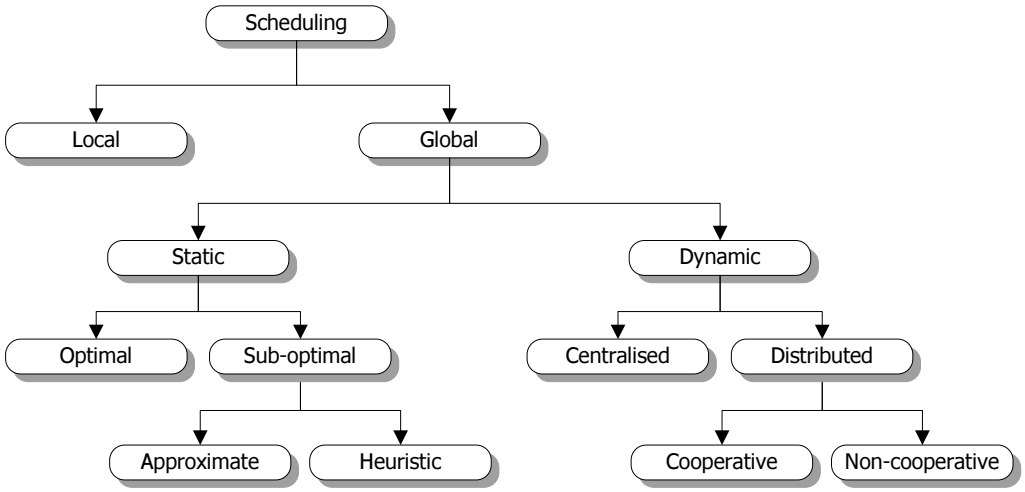


Figure 3.2: A hierarchical taxonomy of distributed systems scheduling approaches, adapted from [58]

The important aspects of Casavant's hierarchy, its applicability to the Grid and its implementation in the current schedulers will be discussed in what follows.

Local vs. Global: Local scheduling is mainly concerned with how the processes resident on a single CPU are allocated and executed. Global scheduling aims to optimise the allocation of tasks among multiple processors, and the Grid scheduling clearly falls into this category.

Static vs. Dynamic This choice indicates the distinction between the flexibility of the schedule. In static algorithms, scheduling is done once and based on the resource and job information available at that time. The scheduler hence requires a "global view" of the resources and an anticipated run time behaviour of the application on which to base its decision - information not readily available in the highly distributed Grid environment. Regardless of the possible changes in the state of the Grid or job queue, no re-scheduling is done. This causes problems if a compute node or a communication link fails. To alleviate these issues, static algorithms use job migration (for example Zhang in [59]) and rescheduling techniques (such as the checkpointing mechanism used in Condor [60]), which brings them closer to the dynamic schedulers.

The advantage of the dynamic scheduling algorithms is in that they perform an online load balancing of the Grid resources at the cost of increased complexity compared to the static scheduling. The following approaches have been used by El-Rewini in [51]:

- *Unconstrained First-In-First-Out (FIFO)* maps the job to the shortest queue. This opportunistic strategy is simple but often results in poor schedules.
- *Balance constrained* strategy occasionally reschedules jobs in order to re-balance the waiting queues. In the Grid, however, communication costs can be high and the time it takes to move the job and the data to a new execution node can cancel out any savings made.
- *Cost constrained* approach takes into account the communication or other costs related to the re-balance of the queues and selects the most appropriate strategy.
- *Hybrid approaches* use a mix of the static and dynamic algorithms. They may perform static mapping for parts of the job with deterministic behaviour, or specific QoS requirements, and fall back to the dynamic scheduling for others.

Apart from these more traditional approaches, some Grid schedulers implement dynamic scheduling using reservations or dynamic FIFO priorities. By negotiating resource reservations on platforms supporting them, the scheduler can reduce the uncertainty of resource availability and performance. Dynamically prioritising the jobs in a FIFO queue has also been examined [61].

An open question remains on which metric do these dynamic scheduling algorithm perform the balancing. A queue job count, for example, can be very misleading as it will be shown that the Grid job execution times can vary greatly. Perhaps the best metric would be the estimated total execution time of the jobs in the queue, a value which the author's work may help deliver.

Optimal vs. Suboptimal Due to the NP-Complete complexity of the scheduling problem, all of the algorithms will generally find suboptimal solutions.

Approximate vs. Heuristic Approximate algorithms require a function to evaluate the solution and a metric to judge its quality. As no suitable objective function existed until recently, no approximate algorithms were developed. A new objective function (the Total Processor Cycle Consumption proposed by Fujimoto in [62]) may help develop new approximate scheduling algorithms.

Heuristic approaches make assumptions on the state of the resources and the job requirements, and then proceed to offer a "reasonable" solution. These algorithms are based on the real world experience and simulations, and since they

are not restricted by the formal assumptions can be more flexible and adaptive. Another advantage of the heuristics is their ability to deliver an acceptable scheduling solution in short time and with a limited computational complexity.

Distributed vs. Centralised High level Grid scheduling can either be done by a single scheduler, or be distributed among several scheduler instances of the same or different type. Centralised approaches, used in all commercially deployed Grid schedulers presented in Section 3.1.3, are easier to implement, but may prove to be performance bottlenecks and single points of failure [63]. Distributed Grid schedulers, are largely still at the research stage (examples in [64, 59]) alleviate these problems at the cost of the deployment complexity and an increased communication cost.

Cooperative vs. Non-cooperative Scheduling nodes in a distributed approach have a number of strategies available to satisfy their scheduling benefit function. In a cooperative strategy each Grid scheduler has its own responsibility but is working toward a system wide goal. Independent or competing strategies allow each scheduler to pursue and maximise its own scheduling benefit function.

Apart from this hierarchical classification of Grid scheduling algorithms, other important aspects and algorithm differences remain outside the scope of this taxonomy. Some of the important differences in how the scheduling algorithms deal with the specific case of the Grid computing and the unique issues it raises, will be discussed in the following sections.

Objective Functions

The scheduler has a higher level objective than simply producing an application to resource mapping: given two valid schedules, it will select one that maximises a certain "'benefit" criteria of the system. What this benefit is, and who defines it, varies according to the point of view. Users submitting their applications to the Grid would like to see their jobs finished as soon as possible, or if there is a cost associated with the job execution they might want to minimise it. The Grid operators, on the other hand, may want to maximise the resource utilisation or the profits from running the Grid jobs. These objective functions are often opposing and competing, and it is down to the scheduler, or the pricing policy in the context of the Grid economy, to make the appropriate trade-off.

The makespan optimisation is almost exclusively used in today's production schedulers. With the emergence of the Grid economy models [30, 34], the scheduler may be asked to minimise the cost at which the computation is done. The problem becomes more complex with the compound functions of these two metrics (makespan and money) where the scheduler must normalise them to judge the fitness of a certain schedule. Some recent research work by Das [65] explores

the use of some of the real world models, such as auctioning, in judging the relative monetary value of a given reduction in makespan and vice-versa.

The objective functions maximising the resource utilisation or the throughput of the jobs are favoured by Grid resource owners and operators. These are often at odds with the application-centric objectives, and schedulers are required to balance these opposing requirements according to some administrative policy. A commercial Grid operator may also be interested in increasing the economic profit extracted. Considering that the Grid offers computing on a service-based model, the quality of service offered to the users will influence their preference toward a certain cluster, the level of demand placed on it and the profit generated. Possible ways of optimising cluster profitability based on the work presented in this thesis are formulated in Appendix C.7

Scheduling Adaptivity

Schedule adaptation is a process in which the scheduling decisions are based on the information, algorithms and the parameters which change dynamically reflecting the past, current and future state of the Grid environment. The need for the scheduling adaptation comes from the heterogeneity of the Grid resources and applications, as well as from the resource performance fluctuations caused by their non-dedicated use and probabilistic availability. The adaptive scheduling algorithms can also be divided according to the source of fluctuations they handle into the following three categories:

Application adaptation algorithms are usually based on the profiling and instrumentation of the source code of a specific application, and profiling of the target platform on which it is to be scheduled. As a result, this tightly coupled approach is not portable or universally usable. This limitation was addressed by Dail in [66] by decoupling the application and resource models from the scheduling framework. Application adaptation through resource reservation was presented by Aggarwal in [67], while Wu in [68] presents a self-adaptive scheduling algorithm that relies on the long-term performance predictions introduced in [69, 70].

Resource adaptation algorithms are concerned with selecting a subset of the resources from the available pool in order to minimise the communications costs between them, achieve high performance, or reduce the performance variability, for example. In a globally distributed cluster such as the Grid, intelligent and application-specific resource selection can greatly increase its performance, especially in the case of the data intensive scenarios [71]. In [66] Dail groups the resources in disjoint subsets according to the network delays, which are then further ranked according to the memory size and the computational power. Subhlok in [72] gives an algorithm to jointly analyse the computation and communication resources for different application demands.

The main challenge of the resource adaptation algorithms is in collecting up to date monitoring data on the dynamic properties of the resources (such as the available network bandwidth, memory, etc.) without excessive communication or storage costs. Often, these goals are achieved by transmitting very basic, compressed metrics such as the last, average or the maximum values for a relatively large sampling period. As good monitoring information is essential for building a representative statistical model and making good forecasts, Section 3.4 surveys the current approaches and discusses the open issues on the topic of Grid resource monitoring.

Performance fluctuation adaptation algorithms aim to reduce the impact of the variable performance levels delivered by a resource and their probabilistic availability caused by their autonomy and non-dedication. Generally applicable, the rescheduling algorithms (in GrADS [73] for example), adapt to the performance or availability drops by re-submitting whole jobs onto a different execution node. In the specific case of the divisible jobs their constituting tasks can be dynamically assigned to the resources as appropriate at the time of execution [74]. Important Grid application classes such as the master/worker, parameter sweep or the data stripe processing can be scheduled in such a way. Previously mentioned application checkpointing algorithms can also be used to reschedule even atomic jobs by generating an occasional snapshot of their entire state and migrating them as necessary.

Non-traditional approaches

New scheduling approaches have been inspired by the Grid's similarity with nature and human society. Both environments are made up of a large number of autonomous entities which are self-ruling but interacting, competing for scarce resources and adapting their behaviour to current environment conditions. Cross-disciplinary problem solving methods briefly introduced here found many applications in Grid scheduling research, depending on how their original problem space was mapped onto the Grid.

Economy models assume a limited supply of Grid resources for which a number of consumers (users or applications) are competing for. Depending on the approach taken, resources are available at a certain cost, may be of a defined quality, or a varying level of community trust [75]. The scheduling process is then seen as the interaction of the resource buyers and suppliers in some mode of market behaviour such as bargaining, open bidding, auctioning or similar. In [30, 31] Buyya applies these economic models to optimise the Grid scheduling, while in [32] same author introduces a novel deadline and budget constrained algorithm that considers the makespan and the cost of the job simultaneously. Economic treatment of the scheduling problem raised other interesting approaches, such as a tender model for Grid applications suggested by Ernemann in [76, 34], and a job

prioritisation model for the traditional schedulers based on the job's committed budget by Zhu [77]. A game theory [78] approach was considered by Young in [79] and was able to find close to optimal solutions in many cases.

Genetic algorithms [80] have found their application as powerful heuristic methods used to find sub-optimal solutions to large combinatorial problems of the Grid job scheduling. They are often combined with other search techniques based on the real-world processes, such as the simulated annealing [81], to avoid locking into suboptimal local solutions. Examples of the genetic algorithms in the Grid scheduling can be found in [82, 83, 84, 85].

3.1.3 Grid Scheduling Implementations

The Grid scheduler and job manager landscape is highly fragmented and utterly confusing. Many implementations can be used as stand-alone solutions, or as part of a layered Grid resource management. This section gives an overview of the most commonly used schedulers on the production and research Grids. The classification is based on their use of the predictive techniques, historical information or the application instrumentation.

Non-predictive

The majority of the commercial schedulers do not make any independent assumptions on the length of the job execution or its resource utilisation. These approaches focus on delivering high-throughput, stable and as deterministic as possible scheduling, often employing fixed prioritisation as means of indicating the relative job urgency.

Condor-G [60, 50] is a high-throughput, policy controlled batch scheduler based on a master-worker approach. It can be used as a standalone system, or as a local job manager for the Globus toolkit with which it communicates using the GRAM protocol. ClassAd [86] language is used to match the application requirements to a suitable hardware. Condor supports job checkpointing, provides node security by using the sandboxing and I/O redirection, and has an integrated monitoring and management suite called Hawkeye [87] (see Section 3.4.4).

Condor's strongest point is in extracting unused cycles from a highly heterogeneous and non-dedicated resource pool, and the ability to migrate and resume jobs during runtime. The scheduling however is FIFO based, with coarse grained prioritisation, and the framework leaves little room for integration of the predictive elements.

N1 (Sun) Grid Engine [88] is an enterprise focused cluster scheduler based on a master-slave agent model that supports a wide range of operating systems and hardware. It can function as a standalone system, or as a local job manager

in a Globus environment. The N1 Grid engine supports parallel jobs, basic resource reservation and job prioritisation. The submission of jobs is through a single master node and each slave runs an agent responsible for task launching, monitoring and reporting. The resource discovery is built-in, but can be extended (for example by using JXTA [89]), and supports the building of complex selection queries.

Scheduling in the N1 Grid Engine is based on a policy and priority modified FIFO model. The role based authentication system can support groups with different priorities, resource reservations and billing options. Multi-site job submission is possible using the Globus Toolkit v3, Grid Engine and JOSH [90]. Manual scheduling to a deadline is possible if reservations are used, but only for applications with known runtimes. The scheduling process cannot readily support deadline scheduling or application run time predictions.

EASY scheduler [91], developed specifically for scheduling parallel jobs, was the first FIFO system to use the "backfilling" method. On job submission, users are asked to specify the number of processors requested and the maximum wallclock execution time for the job. The queuing proceeds in a first-come-first-served manner until a job requests more CPUs than are currently available in the cluster, effectively blocking the remaining queuing jobs from execution. The EASY scheduler examines the running queue and establishes the latest time at which enough CPUs to serve the queued job will become available. It than looks further down the queue and allows execution of jobs requiring less processors to execute if they will not push back the start time of the blocked job - effectively filling in the gaps created by large jobs with smaller ones.

EASY scheduler has been extended to work with other scheduling systems, such as the LoadLeveler [92], and the backfilling method, shown to be fair and efficient, was adapted for use in many later schedulers. However, the dependence on the users for the job execution time estimation makes this approach unviable for many modern applications.

Portable Batch System (PBS) [93] is a widely used batch scheduler in large institutional clusters, and is another example of a centralised master-worker model. Used on its own, it functions as a workload management suite, while integrated in a Globus environment, it serves as a local scheduler and job manager. PBS supports resource reservations, cross-cluster job execution through user mappings and job recovery through rescheduling. PBS is best suited to a well managed and controlled environment, with (mostly) homogeneous hardware and software, and with unified accounting and administration policies.

The scheduling component in the PBS is separated from the job submission server, and through the use of PBS APIs can be modified to implement different scheduling algorithms. The Scheduler communicates with the Server to obtain

submitted job information, and with the PBS resource monitor to acquire the resource utilisation data. It can operate on single or multiple queues and create schedules based on site policies, priorities and the utilisation state of the cluster. Preemptive execution and backfilling are supported, but scheduling to a deadline is not possible. Although it may be feasible to develop a custom PBS scheduler making use of the job runtime predictions, no such effort to date is known to the author.

Load Sharing Facility (LSF) [94] is a popular commercial scheduler geared towards the high computational demand industries such as the financial services and life sciences. Details of the underlying technologies in LSF are not widely available, only a single published paper by the scheduler's author Zhou from 1992 gives some early algorithms [94]. The product literature states that the core of LSF is a virtualisation engine that manages the supply of resources, increases their utilisation and improves the application performance. According to the company web site *"an element of self-management has been built into Platform LSF to offer guaranteed zero downtime, self-adaptive dynamic allocation of resources, and self-healing to reduce management overhead"*.

Platform LSF offers a comprehensive set of scheduling policies with support for fair-share, preemptive and service level agreement based scheduling with advanced resource reservation. The implementation aspects of these have not been disclosed, making functional comparison with other algorithms impossible.

Maui Cluster Scheduler [95] (and related Moab Grid Suite [63]) is a high level Grid meta-scheduler compatible with the PBS, LSF, Sun Grid Engine and other local schedulers and job managers. It supports scheduling policies, dynamic job priorities, resource reservations and fair-share resource allocation. Maui makes a step towards the deadline scheduling by requiring the user to supply an estimate of the maximum running time of a job. This value is used in constructing the initial schedule, which is then further optimised by applying job priorities and an (optional) out-of-order backfilling scheduling algorithm.

Maui maintains the accounting data on the previous user-predicted and actual job execution times, but it does not make any independent forecasts. Analysis done by Maui's developers revealed that the users are likely to grossly overstate the maximum running time of their applications. Even with such unreliable runtime predictions, Maui is able to deliver improved scheduling performance, stressing the importance of this data in creating an effective schedule.

Predictive

Predictive Grid schedulers are still mostly used for research purposes or in specialised clusters scheduling scientific software. Although each of the presented

schedulers takes a different approach, they have all been designed to schedule a specific type of applications onto an appropriately specific set of resources.

Application LEvel Scheduling (AppLeS) [96, 97] is a primary example of the predictive Grid scheduling at the application level. It can optimise the schedule for the user's performance criteria, such as the turnaround time, by predicting the execution times of queueing jobs on the target platforms. AppLeS does this by running a modified, recompiled and instrumented version of the user application on a performance profiled hardware (using Network Weather Service, see Section 3.4.3) using a domain-specific scheduling algorithm. Performing best when scheduling parameter sweep and master-slave applications [85, 71], it can deliver increased utilisation and deadline scheduling. However, the reliance on specific, individual, application and resource models makes this approach acceptable only for the high-value niche applications, or clusters of specialised hardware.

AppLeS bears significant differences to the approach taken in this thesis as it requires each application, set of resources and prediction algorithm to be adapted to its scheduling framework and the deployment domain in question. This requires significant effort on behalf of the user, cluster administrator, AppLeS developer and the software provider. A solution developed in such a way is not portable, and may not perform sufficiently well even with minor changes in the cluster composition, network topology or the usage patterns. Nevertheless, AppLeS has shown possible benefits of the adaptive and predictive schedulers, and an obvious need for their development.

Nimrod/G [98] is a Grid incarnation of a scheduler developed to facilitate large runs of parametrised simulations over a distributed set of resources [99]. Using the Globus toolkit for resource discovery, job submission and security, Nimrod/G enables end users to request job completion by a specific deadline and specify a certain virtual budget for the execution. By offering this "budget" metric, Nimrod/G is looking to provide a framework for market based computational economy where such services could be traded [30, 31]. During the schedule generation stage, a sample of the submitted parametric study application is run on the target nodes and used to extrapolate an overall runtime prediction.

In papers published by its authors, Nimrod/G showed good scheduling performance, with good adherence to the requested deadlines [98]. The trial run prediction method lends itself well to the heterogeneous nature of the Grid. However, Nimrod/G is solely aimed at the parametric study applications, whose execution times are very narrowly distributed, and generally independent of the input parameters. By limiting its scope, Nimrod/G is able to utilise simple prediction methods to achieve satisfactory scheduling performance. Although these applications form an important group of the scientific software presently running on

the Grid, a general purpose scheduler must also be able to handle other types of applications.

PACE/Titan toolset [100, 101] is a deadline based scheduler supporting runtime predictions, performance modeling, and out-of-order job executions. PACE [102] component uses the pre-execution modelling to predict the job runtime and the resource utilisation based on the hardware and software characterisation templates, and an evaluation engine estimating the application performance on different resources. It requires all applications to be recompiled with the PACE libraries and all execution hardware profiled so that the performance templates can be made. Titan [100] is a workload management component of the toolset. Using the performance predictions supplied by PACE, Titan uses a genetic algorithm to optimise the execution schedule reducing idle time, makespan or scheduling delay, while maintaining the deadline adherence. The scheduling is dynamic, and is constantly performed on the pool of outstanding jobs, replacing the current best schedule if a better one is found.

PACE/Titan toolset is a good example of the power of predictive scheduling techniques and the challenges of the job runtime predictions. Good results have been reported [103], but despite these the main drawback of the toolset is the need to recompile the applications, and extensively profile the target hardware. For a large number of users running different applications on non-dedicated resources, such as in a typical utility Grid scenario, this may be impossible. The main strength of the PACE/Titan scheduler remains in running the high-end scientific applications on a relatively static pools of high performance dedicated hardware.

ICENI [104] is a predictive scheduler aiming to explore the role and the flow of the job meta-data in the computational Grids. It incorporates a separate scheduling component, job launching framework and a performance repository holding historical data on the job execution times on different architectures. The scheduling component is extensible and supports multiple concurrent and competitive scheduling algorithms (ICENI authors have presented four such algorithms in [79], including the simulated annealing and the game theory methods). The prediction engine treats the applications as a collection of simple components connected as a directed acyclic graph (DAGs, see [105]) with varying depths and dependencies. It introduces a user-defined benefit value, such as the target execution time or the computing cost, which the scheduling process aims to optimise.

ICENI parts from the traditional approach of the batch schedulers and offers predictive, out-of-order job execution and several Grid specific benefit functions. Although the importance of the meta-data is considered, its integration in the overall flow of monitoring information could have been more thorough. ICENI falls short of offering a fully fledged deadline scheduling, but optimisation of the wallclock job execution time can be done using the benefit function. The core

scheduling work focuses on the algorithm development, recognising the need for approaches of varying complexity, but little attention is paid to the job execution time prediction methods, their accuracy and computational cost. Due to an open architecture and modular design, ICENI offers a good platform for deployment of third party components and their testing in a production-like environment.

3.1.4 Summary

This section has presented the general scheduling problem, as applied to the parallel and distributed computing systems, and some unique aspects of the Grid platform which pose specific challenges to the legacy scheduling approaches. This examination of the broader scheduling process showed that efficient scheduling depends on the good algorithms for resource discovery and efficient access to the monitoring data. Some of these issues were addressed as part of the SO-GRM project and will be discussed in Appendix A.

From an extensive survey of the Grid scheduling algorithms, their complexity, adaptivity and objectives, it became clear that the dynamic properties of the Grid, and its non-deterministic nature, are the hardest problems in the traditional scheduling approaches. Many techniques, which would be better suited to overcoming these Grid specific issues, would require estimates of the execution times of the queued jobs in advance of their start.

With the transition of the Grid paradigm into a service-orientated infrastructure, the objectives of the commercial Grid operators and end-users diverge. Emerging new concepts, such as the Grid economy, are seen as ways of optimising the objective functions of both the users and operators. Delivering Grid scheduling with the deadline and budget constraints will depend on the sound economy models, and the ability to predict job execution times.

The section has also presented numerous implementations of the Grid schedulers, separated into two categories: those that in some way try to predict the execution times of the submitted jobs, and those that do not. The number of the predictive schedulers, and the numerous ways in which they attempt to anticipate the job execution times strongly motivate the author's further work.

3.2 Performance Predictions

Forecasting is a process of estimation in unknown situations [106], and is used extensively in support of decision making. In the following, it will be used interchangeably with a more general term "prediction" which is usually associated with forecasting time-series data.

This section will discuss the problem of forecasting the computational load and the resource performance in the context of the distributed deadline scheduling. It will survey the current methods and approaches used in forecasting the job execution time, state the particular issues and challenges of making such predictions in the Grid environment, and briefly discuss the significance of the outlier data points and other "anomalous" workload properties.

3.2.1 Problem Statement

From the survey of the scheduling algorithms and implementations in the previous section, it is clear that significant performance and functionality improvements could be achieved if an estimate of the job execution time on a given resource can be made. Therefore, the problem is one of delivering runtime forecasts of sufficient quality, and based on the available information prior to the execution of the submitted job.

Related Forecasting Problems in Distributed Computing

Similar problems abound in the management, provisioning and planning of the distributed computational resources. Attempts were made at modelling and predicting many performance influencing, dynamic, properties of these systems such as:

- Host and CPU load by Dinda [107] and Lingyun [108]
- Queue waiting time by Downey [109]
- Network available bandwidth by Wolski [22, 110, 21] and the file transfer time by Vazhkudai [19]
- Resource discovery performance in the Grid Information Systems by Keung [111]

Despite the diversity of the topics listed, all of these approaches rely on several common forecasting methods that will be discussed in Section 3.2.2.

Challenges of Job Execution Time Estimation

The complexity and the quality of the runtime predictions is proportional to the volatility of offered load and the variability of the service rate. In embedded, robotic, or industrial control applications for example, sensor events are serviced by processes with known execution time, usually running on real-time operating systems and hardware [112, 113, 114, 115]. Adherence to an execution deadline is then guaranteed by the deterministic nature of the system and all of its components.

Grid computing is a much more probabilistic environment in which both the computational load and the hardware service rates vary. The apparent randomness of the human behaviour, the primary source of the computational load in the Grid, leads to variable service request rates. In addition, the submitted applications vary greatly in terms of complexity and resource requirements, and their execution time is often dependant on the parameters of the specific run (for details see Chapter 4). At the same time, the autonomy of the Grid resources means that their availability is not guaranteed, and their non-dedication implies fluctuating service levels offered to the Grid applications. In these circumstances, estimating the job execution times becomes a real challenge.

3.2.2 Prediction Approaches

This section will describe the approaches used in the current research work and implementations for predicting the job execution time. These methods are not mutually exclusive, and are often combined to yield an increased prediction accuracy. The focus of this discussion is on the body of related research, but references are provided to production schedulers based on the mentioned research work.

User Provided Estimates

The simplest and the oldest approach to acquiring the job runtime predictions is asking the user to give an estimate. The reasoning behind this method is that the user submitting the job knows it best and would be able to somehow judge the level of computational complexity requested from the application. The user is also presumed to have the benefit of some historical hindsight and can make an educated guess based on the previous application runs in similar conditions.

User's estimates are communicated to the scheduler either implicitly (by submitting the job to a queue with a certain maximum execution time) or explicitly (by stating the maximum or estimated execution time as a parameter to the scheduler). The former was very common in the legacy batch systems and is still widely used today (in some versions of PBS scheduler, see Section 3.1.3), while the latter can be found in the more recent Grid schedulers (such as Maui [95]). Whether, and under which conditions, will the job be hard limited by the given maximum execution time, or whether it will be allowed to continue execution past its declared maximum runtime, or the limit of the queue to which it was submitted, is subject to the scheduler implementation and the local policies.

The simplicity of this prediction method is appealing, and has worked on the previous generations of the time-shared, high performance systems where the resource had deterministic performance and the users were repetitively submitting specialised applications. In the Grid context however, users are not aware of the constantly fluctuating performance levels of the execution nodes, and may not even have an in-depth knowledge of the application they are running. This leads

to extremely inaccurate job runtime predictions documented, amongst others, by Lee [116] and Downey [109]. Another possibility, given the conflicting interests of the users and the Grid scheduler is the manipulation of the scheduler by the users wishing to "jump the queue" by intentionally giving lower runtime predictions.

Application Instrumentalisation

Application instrumentalisation enables the resource management middleware to gain an inside look into the functional, performance influencing, components of the application. The method augments the core problem-solving source code with an additional functionality that can, depending on the implementation, passively analyse the application performance, estimate the required resource utilisation, predict time to completion, or actively adjust the speed of the execution. The process of instrumentalisation involves significant changes and recompilation of the user's application, and profiling it on all of the target execution platforms.

The research in this topic has focused on the best ways to capture the internal organisation of an application and discover its performance influencing parts. To this end, the directed acyclic graphs (DAG) [117] have been frequently used [118, 119]. Object oriented methods have been proposed By Gergeleit in [120], while the most notable implementation remains the AppLeS scheduler (see Section 3.1.3).

The main benefits of the application instrumentalisation method are its high prediction accuracy, and the ability to estimate the job time-to-completion used in deciding whether to reschedule a running job elsewhere. The need for source code changes and recompilation is a major issue as, even if the code is publicly available, the process is a laborious and expensive one. The approach is very specific to the software and hardware in question and hard to adapt to a general purpose utility Grid. Application instrumentalisation is therefore best suited to specialised clusters running high value niche applications.

Application and Hardware Profiling

Profiling approaches use a variety of algorithms to capture the dynamic behaviour of the applications and the hardware in a model suitable for prediction generation. This approach is similar, and often used together, with the application instrumentalisation. Profiling, however, does not require alterations or re-compilation of the source code. It rather tries to create the model non-intrusively, passively analysing the applications and monitoring the hardware. A successful profiling technique will generate models in an (semi-)automated way that can evaluate different performance scenarios and capture the system's properties with the least number of parameters.

The profiling of software can be done using the test runs of sample code on the target hardware [98], creating the system logic models [121], or using the binary

code analysis [122]. Hardware platforms are most often described by their static properties (such as the amount of installed memory, the speed of the CPU, or the FLOPS rating), or by using a real-world application benchmark (such as the SPECmark [123]). More detailed analytical models [121] can also be developed, usually for more specialised systems.

Most of the cited algorithms in this category produce very accurate predictions. The applicability of the approach however, still remains limited. Most of the profiled hardware is monolithic and dedicated, the properties which do not readily apply to utility Grid clusters. The modelled applications are highly specialised, well studied and often performance deterministic with a narrow runtime distribution (such as the parameter sweep application scheduled using Buyya's Nimrod/G, see Section 3.1.3). The modelling method itself, while requiring less involvement than the full instrumentalisation, is still not fully automated and usually requires the involvement of the Grid administrator and the end-user. Overall, application and hardware profiling serves as a good starting point for the development of a more automated and generalised approach based on application templates [96].

Statistical Methods

If the successive historical job execution times are collected by the Grid middleware, then these can be analysed using the statistical analytical methods in an effort to predict the future job runtimes. Experience has shown that even some seemingly random or very noisy series (such as the stock prices or the commodity demand) can be modelled and predicted to a usable error margin [124] using statistical methods. Rather than trying to capture the cause of the application's achieved performance, these methods model the end effect (the job runtime) directly. The following statistical methods are most frequently used in the prediction of the job execution time or the closely related performance metrics.

The mean and the median based methods [125] are often used due to their (computational) simplicity. They are frequently used and reported [109, 126] as they form a benchmark for other, more advanced, statistical methods. The mean and the median based forecasts are very dependant on the distribution of the data points and the approximation used to represent them.

Regression techniques [127] attempt to model the relationship between the execution time and another variable, or in the case of the auto-regression between the current and the lagged historical values of the execution time itself. These methods are extensively used [107, 128, 126, 129] due to their predictive power, and the ability to capture cyclic behaviour.

Moving average methods [130] compute the weighted average over a number of historical values of the modelled variable. They are often used in conjunction with the regressive techniques [131], but require a non-deterministic time to fit.

Stochastic values [132] are ranges of values which can be represented using different distributions, intervals or histograms. They are able to communicate the dynamic properties of the system better than the spot values, and capture more information on the variability of the modelled metric. Stochastic prediction methods have been used by Schopf in [133, 134, 135].

Homeostatic and tendency based methods are based on a relative value of the last historical data point. The homeostatic strategy assumes that if the current value is grater that the historical mean, the next value is likely to decrease. The basis of this approach is that the data will be "self-correcting" or so that it will return to the series mean value. A tendency based strategy states that if the last sample was of increasing value the next one will be too. An important source of error is the inability to predict the "turning point" when the series changes direction. These methods have been adopted by Lingyun in [108]

3.2.3 Special Events Detection

When using statistical forecasting techniques, the quality of the predictions will greatly depend on the variability of the data and the presence of outliers, anomalous data points or high-frequency components. Anomaly detection and filtering is a large research topic on its own, with a range of applications from seismology to medicine. In the context of the distributed systems, it is most often applied to the network monitoring and management.

A small body of published work on analysing anomalous behaviour in the workload traces goes as far as identifying and acknowledging the presence of outliers both in the job execution time data and the job arrival rates. Tsafrir has shown in [15] that these can have significant effects on the scheduling performance and suggests ways of filtering them out of the dataset.

Further discussion of the statistical properties of the Grid workloads, together with the merits and problems of excluding the anomalous data points is deferred until Chapter 4.

3.2.4 Summary

With the increased research interest in the deadline scheduling, and other alternatives to batch scheduling, the ability to forecasting the execution time of the queued jobs is seen as a necessary functionality. Delivering such predictions, in the context of a general purpose utility Grid system, proved to be difficult.

The discussion of the forecasting methods currently used reveals that the runtime predictions supplied by the Grid users are unreliable, and that the application instrumentalisation and modelling techniques yield good results but require source code changes or extensive and preemptive analysis of the hardware and software. In a dynamic environment like the Grid, these are seen as prohibitively high costs.

Statistical forecasting methods have a potential to deliver job runtime predictions in an automated way, transparent to the user and easily manageable by the administrators. Although the initial prediction accuracy may not be on a par with some more complex methods, further algorithm improvements and careful handling of the outlier data points could significantly increase the prediction accuracy.

3.3 Workload Characterisation

To successfully select and apply a performance forecasting model, good understanding of the statistical properties of the workload are needed. The topic of workload characterisation has been extensively researched before, but little work is evident in the context of the Grid computing.

Since the Grid architecture is significantly different from other distributed and parallel systems, one can expect that the workload will also be significantly different. A close examination of its properties is therefore warranted. This section will first give a brief historical overview of the workload characterisation, followed by the discussion of the important workload metrics and their treatment in the literature.

3.3.1 Historical Overview

Knowing the properties of the demand that will be presented to the system is crucial in its planning, performance tuning and bottleneck optimisation. Previous workload studies have dealt with workload characterisation of interactive [136, 137] and database [138, 139, 140] systems, communication networks [141, 142], and Web services [143, 144, 145] amongst others.

Although often difficult, characterisation through proper analysis of the real-world data is important in avoiding flawed system designs [146, 147]. Analysis of the actual Internet traffic patterns by Leland [148] and Paxson [149], for example, led to the ground breaking departure from the Poisson-based model and had significant impact on many other workload models, such as those of the Web server performance [150, 151, 144, 145].

Historically, the focus in the workload characterisation of the distributed and parallel systems was on developing models able to generate representative traces to be used in other simulation work [23]. With the recent interest in predictive

scheduling, some researcher have started to examine workload properties looking for possible forecasting models [152].

3.3.2 Modelling Scope

As the computational workload consists of several layers of jobs, tasks, routines and instructions, so can workload models be focused on capturing the properties of one or more of these layers. One option is to model these levels explicitly, creating a hierarchy of interlocked models for different levels, while another is to study them as opaque boxes and model their response to input data.

Work by Calzarossa and Sarazzi [153] established the foundations for modelling the processes generating the workload. Their methodology subdivides the workload per each user, identifies similar commands using clustering techniques and chooses several representative ones. It then describes user behaviour through probabilistic User Behaviour Graphs [154] and Markov chains [155, 20, 156], and uses aggregation-disaggregation techniques [153] to obtain the global model parameters. This process captures both static and dynamic properties of the intrinsic workload generation process in a concise form and can be used to generate representative workload traces.

This detailed approach to workload modelling quickly becomes overcomplicated, and while able to generate good representative traces, it does not provide much insight into the statistical properties of interest to the job execution time predictions. The remainder of this section will therefore focus on past work using statistical techniques to directly characterise workload's general properties.

3.3.3 Workload Properties

Previous characterisation studies of parallel systems model a number of workload properties relevant to predictive scheduling techniques. The arrival process, the job's requested, queueing and execution times, and the degree of parallelism are some of the workload's most studied aspects. The following overview of the findings and the related work is based on the job traces from pre-Grid clusters; only the most recent work by Li [26], Iosup [157], Medernach [27] and Dobber [28] are based on the actual production Grids and will be treated separately in Section 7.1.

Arrival Process

The daily fluctuation of the number of submitted jobs, and its correlation with the human work pattern has been universally reported [23, 24, 26, 27, 152]. Most traces show a differentiation between the weekday and the weekend arrival rates, except as reported by Cirne in [23], and some authors choose to ignore the lunch hour dip [24]. Arrival distribution function has been modelled using 8^{th} –

12^{th} degree polynomials [23], log-normal [24], (hyper-)exponential and (hyper-)Gamma [24, 26], and Weibull and Pareto [26] distributions.

Although the majority of the previous work has assumed Poisson distribution of the inter arrival rates, Medernach has reported strong burstiness at all time scales [27]. Cirne has studied the link between the job arrival time and other job properties, including execution time, but could not identify any statistically meaningful correlations [23].

Job Execution Time

In his work, Gibbons assumes normal distribution of the job execution times [152] and approximates it with the Student's t distribution. Cirne and Berman derive the job runtimes from the user job requested time and the explicitly modelled accuracy of such user predictions [23]. Weibull, log-normal and Gamma distributions were used by Li in [26], while Lublin fits a hyper-Gamma function [24].

All authors report a very wide range of the execution times and often remove outliers (usually past 95th percentile) or perform logarithmic transformations.

Job Request Time

Job request time is specified to the scheduler by the submitting user, and is an indication of the maximum expected length of execution. The relationship with the actual execution time remains contentious: Cirne [23] has used it to derive the job run time, Li [26] has found it strongly correlated with the actual runtimes, but many other authors, including Medernach [27], Lee [116] and Tsafrir [158] have found this information to be highly unreliable.

Queue Wait Time

Queue wait time, or the time that each job spends in the scheduler queue, is an indication of the scheduler fairness and prioritisation policies. Medernach has reported wide variation between wait times for different groups of Grid users [27], but this metric has not been studied in greater detail.

Job Parallelism

The number of the nodes or processing units (CPUs) used by the job simultaneously and in parallel has been reported to have a strong preference for the power of 2 values (2, 8, 16 etc.) [23, 24, 26]. Through direct interviews with the users, Cirne has confirmed this to be due to behavioural inertia [23] as most legacy systems could only support power-of-2 parallelism. Job size has been modelled using log-normal distribution [26, 23], but with limited success. Although intuitively job execution time should be inversely proportional to the degree of parallelism, opposite [24] or no correlation [26] has been found.

Memory Usage

Overall, the memory utilisation was reported to be low and highly modal [26], a factor attributed to the use of standard dynamic libraries. As dynamic properties of memory allocation are lost due to the way data is collected, the value of this metric is significantly reduced in the context of predictive scheduling despite its strong correlation to the job runtime reported by Li [26].

Cancelled Jobs

Many workload traces contain a large proportion (up to 23%) of cancelled or unsuccessful jobs [26, 23]. Cancellations are either due to the user actions, or the failure of the job while setting up its working environment (missing files or libraries, inadequate resources etc.). The cancellation rate has been modelled using the log-normal [26, 23], hyper-exponential or Weibull distributions [26].

User Behaviour

Both Lublin [24] and Li [159] acknowledge the strong influence that the user's habits and behaviour patterns have on the characteristics of the workflow, but do not investigate this further. These two authors also make passing remarks on the evolution of the workload through time and propose that further studies of this effect should be undertaken.

3.3.4 Summary

Many aspects of the planning, provisioning and management of computing systems are strongly influenced by the service demand that will be presented to it, thus making the characterisation of such workload an extensively researched area. The majority of these previous studies have used older traces collected from parallel clusters in the 1990s which, due to some specific properties of the Grid, are not very representative of the modern, highly dynamic, distributed clusters.

This section has provided the historical overview and the scope of the previous workload characterisation studies. It has also outlined the studied metrics and the reported findings on their properties. It also reiterates the important distinction between the previous studies whose purpose was to capture the properties of the workload that will enable the generation of similar, statistically valid usage traces, and the one undertaken by this thesis which was aimed at supporting the selection and the implementation of predictive algorithms.

3.4 Grid Monitoring Tools

The previous section has underlined the importance of the quality, timeliness, and the accuracy of the Grid job and resource monitoring data in the workload

characterisation process. This section will briefly present the most often used Grid monitoring tools and discuss their strengths and weaknesses. A quantitative study of their performance can be found in [87].

Some of the issues identified in this survey have been addressed by the author through an improved monitoring system presented in Appendix A.

3.4.1 Ganglia

Ganglia [160] is a hierarchical, distributed, monitoring system using XML for data representation and round-robin fixed size databases* for storage. Ported to a wide range of hardware and operating systems, and deployed on the production clusters containing over two thousand nodes, it has proven to be a stable, robust and scalable system with low overheads.

Ganglia monitors can track both dynamic (current CPU load, available memory) and static (machine architecture, OS version) host properties; custom metrics can also be added. The cluster nodes running Ganglia can either publish their measurement data, collect data published by other nodes, or do both thus creating a distributed data repository. Low overhead communication is implemented through broadcast messages within the cluster, or unicast links between the clusters. A convenient web-based visualisation package is also provided.

One of Ganglia's primary strengths, the fixed sized databases, is also its main weakness in the context of workload characterisation and job runtime predictions. In the round robin databases, collected monitoring data is periodically consolidated (using simple functions like average or min-max), leading to an irrevocable loss of the high frequency detail and the alteration of statistical properties. A method for solving these issues is proposed and implemented by the author in Appendix A.

3.4.2 Relational Grid Monitoring Architecture

R-GMA [49] is a web service implementation of the GMA specification [161] providing access to the monitoring information through a relational database concept. GMA standard recognises that the performance monitoring information differs from other forms of system or program-produced data: it has a short lifetime, is frequently updated and is stochastic in nature [87].

GMA monitoring architecture consists of three components: data producers publish their capabilities in the directory, and provide information directly to the data consumers based on their subscription to the particular information feeds. Such approach implies a separation of the meta-data describing the monitored metric and the stream of the actual measurement data. Relational GMA system builds on this model by implementing the producer – consumer communication (and the directory functionality) through a relational database.

*see http://oss.oetiker.ch/rrdtool/

Grid Monitoring Architecture provides a bare framework for which adequate information providers and consumers need to be developed. Although the whole Grid community would benefit from its wider adoption, few installations use it. The EGEE project [29], R-GMA's biggest proponent, and its monitoring database may contain significant amount of data which could be of great use in understanding the Grid applications and their statistical properties. As with any other centralised approach, the registry and the database schema could be a single point of failure, unless properly replicated.

3.4.3 Network Weather Service

The Network Weather Service (NWS) [22] is a resource monitoring and forecasting system. Since its forecasting of resource performance levels and availability was discussed in Section 3.2, the primary focus here is on its monitoring aspect.

The NWS system architecture [110, 21] is based on four separate components: multiple distributed Sensors, Forecaster, Name Server and Persistent Storage. Although NWS was primarily developed as a network latency and bandwidth monitoring tool, its open interface allows for the addition of third party sensors. A single instance of the Name Server and the Persistent Storage processes is run in the cluster. The Name Server is the only well-known address used by the system, allowing for both data and services to be distributed, but also creating a single point of failure. Data storage is implemented using circular data files.

Sensor implementation in the NWS uses an intrusive measurement approach by running a compute intensive code, or transferring data across the network. While this may reflect poorly on the system loading or network congestion, it does provide the real measure of the performance as experienced by the applications. Network sensors are organised into hierarchical "cliques" performing mesh measurements within these, and point to point measurements between different cliques and hierarchical levels.

Circular storage methods used in NWS are similar to round-robin databases used by the Ganglia Cluster Monitoring, but provide even less historical information. From the workload characterisation point of view, data provided by the NWS is of limited use. Today, NWS is much more known and used as a bandwidth and CPU load forecasting tool, than as a straightforward monitoring system.

3.4.4 Other Monitoring Systems

Several other monitoring systems are used in the Grid community, usually with a more specific focus on one of the aspects of the system's operation. Often, large projects assemble toolkits of loosely coupled, best-of-breed components, and distribute them as a part of their customised Grid middleware.

GridMon [162] is a UK e-Science project monitoring network performance between each of the regional e-Science nodes. GridMon confirms connectivity and measures packet loss, round trip time and TCP/UDP throughput by using simple scripts or sample data transfers. All measurements are done in a mesh between each Grid node, which is generally intrusive and non-scalable. GridMon can publish its measurements using a Web based visualisation suite, LDAP service or an OGSA compliant web service.

Condor Hawkeye [87] is a part of the Condor system (see Section 3.1.3) based on the ClassAd [60, 50] messaging protocol. It configures the Condor pool master to periodically run monitoring scripts and generate appropriate ClassAd messages. Hawkeye leverages a large installed base of the Condor, and requires little administration effort. However, due to the (in)frequency of the measurements, it is more of a summary utilisation and problem reporting tool than a high resolution resource utilisation monitor.

3.5 Grid Simulation Suites

Testing of novel Grid scheduling algorithms and approaches poses a significant challenge: the importance of the hardware federated in the large production Grids prevents running of an untested scheduler, but small Grid testbeds often do not have all the dynamic properties and the diversity of a real system. In those cases, the use of the Grid simulators and emulators is the only remaining option.

3.5.1 SimGrid

SimGrid [163] is an agent based scheduling simulator with support for the realistic Grid topologies imported from the third-party topology generators. In SimGrid, all low level compute and network resources can have variable background utilisation (supplied from the monitoring trace files), and be contended for using different strategies (FIFO, FRFO *, fair share) [164]. Once the simulation scenario and the hardware topology has been developed, different scheduling techniques can easily be implemented and repeatable measurements made to assess their merits.

SimGrid builds on the best approaches from more complex and specific simulators, while maintaining the simplicity and good performance levels. Its use by a number of research projects, and numerous publications of the SimGrid simulated results have confirmed it to be scalable, configurable and extensible enough to simulate a wide variety of scheduling problems [165]. Validation of the SimGrid results remains a difficult question, especially in a relatively new setting that the

*First Ready First Out

Grid is. The problem is alleviated to some extent by the fact that SimGrid is based on the models previously accepted in the scheduling community.

3.5.2 GridSim

GridSim is primarily a scheduling economy simulator focused on supporting the parametric applications studies [166]. It can model the geographical and the social aspects of the Grid environment using variable background resource utilisation based on the time zones, busy hours, or days of the week. GridSim supports the definition of the user's deadline and budget constraints, but is severely limited by the need to specify both the resource performance and the application computational costs explicitly (using MIPS*).

Although based on an already established simulation platform, GridSim is not as methodological in simulating realistic network topologies, link congestion, resource contention, and parallel applications as SimGrid. Poor documentation further mars development of genuinely useful simulations. Despite this being a general purpose Grid simulator, GridSim is targeted at the parametric research applications and economy driven scheduling approach.

3.5.3 MicroGrid

MicroGrid [167] is an online emulator, providing a virtual Grid environment on which real Grid middleware (such as the Globus Toolkit) and Grid applications can be run. It relies on the operating system to provide virtualisation, and external applications (VINT/NSE) to simulate networking events. Computational resources are characterised by a scaling factor to their real performance.

MicroGrid simulator has been validated by the authors in different testing scenarios [168]. The virtual Grid approach is the most realistic one, and of particular interest when the middleware behaviour to events such as node or network failures is of interest. However, the need for global coordination of resources in the virtual Grid enforces a "maximum feasible simulation rate" on the whole environment. Although theoretically possible, large Grid simulations with complex resource pools could be prohibitively time consuming to execute.

*Millions of Instructions Per Second

Chapter 4

Workload Characterisation

All models are wrong, but some are useful

— George E.P. Box, Professor Emeritus

Having surveyed the previous research and related literature on the characterisation of parallel and distributed workload, it became obvious that few have covered computational grids. The key properties of this new kind of distributed approach are substantially different and therefore warrant a thorough investigation. Workload characterisation reported in this chapter uses statistical analysis to study the properties of the load presented to a Grid cluster, the patterns of user behaviour, and the predictability of metrics of interest to the deadline scheduling.

The chapter opens by outlining the scope and the aims of the characterisation study in Section 4.1 followed by a detailed discussion of the analysis methodology given in Section 4.2. Sections 4.3 through 4.6 present the general workload characteristics, its diversity and differentiation based on several meta and temporal properties, correlations between the job execution time and other metrics, and a study of the effects of temporal and sampling locality. The chapter concludes with a summary of the observed behaviour and characteristics given in Section 4.7

4.1 Introduction, Scope and Motivation

As the primary use of our predictive scheduling methodology will be a general use utility Grid cluster, a suitably representative workload trace was required. Most of the early Grid installations were bespoke systems with the purpose of running one, or very few, specialised applications. These systems are well served by the specially focused predictive schedulers, discussed in Section 3.1.3,

as detailed profiling, instrumentalisation and customising is more practicable. In a general purpose utility Grid, however, one can anticipate a wide variety of applications with significantly varying requirements and statistical properties. Characterisation of one such workload is here presented.

4.1.1 Goals

Unlike many similar workload studies [150, 154, 137, 136, 142, 144, 155, 153, 24] whose aim was the generation of new traces with realistic properties, the primary motivation for this work was a deeper understanding of the workload behaviour in order to develop a sound predictive model. In this respect, the question which statistical model describes the workload best was second to the understanding why it behaves in such a way, and what the effect of such behaviour or statistical property will have on the workload predictability and the selection of the forecasting method.

The analysis paid specific attention to the investigation of the following workload characteristics:

- Statistical properties which may influence the selection of the forecasting methods or the analytical approach (autocorrelation, normality of the distribution, presence of long-tails, self similarity, etc.).
- Cyclic behaviour and seasonal variations which can help anticipate future resource demand levels.
- Correlations between the different metrics and between the metrics and the job meta-data that can reduce data variability and increase prediction accuracy.
- Evolution and longer-term changes in the workload which would require dynamic tuning of the forecasting algorithm.
- Presence of anomalies, drastic or sudden changes in the workload behaviour, their impact on the predictability and methods for handling them.

In-depth knowledge of the workload was essential in answering the two key questions supporting the entire predictive approach of the thesis. Firstly, establish the possibility of using the job meta-data to reduce the variability of the observed execution times and thus increase the forecasting accuracy. Secondly, by using the appropriate statistical analysis tools, assess the predictability of the job execution times and indicate the candidate models or distributions suitable for making forecasts.

4.1.2 The UCL Central Computing Cluster (CCC)

The characterisation was done on the data collected from the Central Computing Cluster (CCC) of the University College London's Research Computing facility. The installation went live in September 2004, and with most deployment problems solved by January 2005 the number of users grew quickly (see Table 4.3 on page 75). Table 4.1 gives more information on the installed hardware and software environment.

Hardware Properties	
Number of Nodes	100
CPUs per node	2
CPU Type	AMD Athlon @ 1200Mhz
Memory per CPU	4096MB
Network Interface	Switched Ethernet @ 100MBps
Software Properties	
Operating System	Linux 2.4
Grid middleware	Sun Grid Engine 6.1

Table 4.1: The CCC hardware and software configuration

The user base at the facility was very varied and comprised research groups from within the UCL and from academic and research institutions elsewhere in Europe. The submitted workload presented a mix of research applications from the high energy physics, biomedical, engineering, and other fields.

4.1.3 Data Acquisition

The fact that the CCC facility is in production use and servicing a large portion of UCL's research community meant that only reliable middleware and resource efficient system monitoring tools could be used. Data analysed in this thesis was obtained by parsing the Sun Grid Engine's job accounting file [88] which records an entry for each job executed on the Grid containing around 50 essential job metrics. The benefit of this approach was that it is based on a passive monitoring technique and requires no additional software to be installed. It is, however, inflexible in the number of the job properties recorded and the way they are collected.

The accounting file does not contain any auxiliary system data, and due to administrative practices at the site, it would be very difficult to correlate the workload features with the external events such as power failures, cluster downtime, or system maintenance.

The job accounting data was parsed to produce a comma separated file containing a single line for each submitted job. The fields collected are described in Table 4.2.

Metric	Description
Job_ID	Unique, serial integer number assigned to each job by SGE. Non-continuous due to the jobs removed from the queue before execution
Owner	Anonimised, numerical integer identifier of the UNIX username submitting the job
VO	Anonimised, numerical integer identifier of the Grid Virtual Organisation submitting the job
Hostname	Anonimised, numerical integer identifier of the worker node executing the job
Job_Name	Anonimised, numerical integer identifier of the executable run or the shell script invoked
Sub_Time	UNIX epoch time of the job submission
Start_Time	UNIX epoch time of the job starting execution
End_Time	UNIX epoch time of the job ending execution
WClock	Wallclock, or real time, the job has been executing. Also equals End_Time - Start_Time
CPU	CPU time used by the job, as reported by UNIX */proc* file system
Mem	Total amount of memory allocated by the job, as reported by the UNIX */proc* file system

Table 4.2: The CCC accounting file fields and their description used in the workload characterisation study

The characterisation also looks at another, derived, metric which is helpful in understanding the workload. Wait time, the time spent by the job in the scheduling queue, is calculated as the difference between the Start_Time and the Sub_Time.

Although the Sun Grid Engine supports parallel environments, the properties of the environment requested by the user were not recorded in the accounting file. This meant that while it was possible to establish that about 1% of the jobs requested multiple CPUs no further analysis of the effect of the parallelism on their execution was possible.

The data analysis and plotting was primarily done using MathWorks Matlab R14* with the Statistics Toolbox. Where non-standard, or custom built, Matlab functions were used, appropriate references will be given.

4.2 Specific Methodology

The need to analyse the extensive sampled data, concisely report the findings of the characterisation study, and formulate meaningful and statistically valid hypotheses as the basis for further work on predicting the job execution times required a substantial methodological preparation. This section will begin by

*see http://www.mathworks.com

outlining the higher-level approach of the exploratory data analysis, and continue with the presentation of the methods used for describing the value distributions and measuring their location and dispersion. The notions of scale invariance and self-similarity will be introduced and the tools used for establishing the cyclic behaviour, correlation and temporal locality of the job properties will be presented.

4.2.1 Exploratory Data Analysis

Exploratory data analysis (EDA) is an approach to data analysis, first suggested by John Tukey in 1977 [169], that employes a variety of, mostly graphical, techniques to maximise insight into a data set, uncover its underlying structure, extract important variables, detect outliers and anomalies, and test underlying assumptions [170]. The distinguishable feature of this method is that it postpones the usual assumptions about the model that can be used to fit the data, thus allowing the data itself to reveal its underlying structure. EDA has established itself more as a "philosophy" of how to dissect a data set, what to look for, how to look and how to interpret the findings.

Exploratory data analysis techniques are graphical, with only a few numerical methods. The reason is that by its very nature, the role of the EDA is to serve as a tool for an open-minded exploration of the data. In combination with the pattern-recognition humans possess, these graphical tools are the best way to reveal new, often unexpected, insights into the data. Typically, EDA makes no assumptions of the nature or properties of the data being analysed, but uses it as a "window" for looking into the core process that has generated it and will most likely continue to generate it in the future.

The ultimate goal of the exploratory data analysis is therefore to gain a real insight into the properties of the data set and its underlying structure, while at the same time providing all the specific items needed to properly handle the data. These items include a good-fitting model, estimates of the model parameters, a sense of the robustness and variability of the data, a list of factors influencing the process and conclusions whether the influence of those factors is correlated and statistically significant.

EDA has established itself through several seminal publications [171, 172] as one of the major data mining and analysis approaches. However, it can be misused leading to a systematic bias problem if the same data is used to suggest and test the same hypotheses. Appendix B presents the author's attempt to avoid such mistakes by undertaking a characterisation study of an additional Grid workload.

The following will introduce the tools commonly used in the exploratory data analysis such as the scatter and box plots, normal probability plots and other EDA techniques.

4.2.2 Value Distribution

Based on the EDA principles, the statistical graphics will be used extensively throughout this chapter. Their advantage is in the lack of any underlying assumption about the sample statistics, ability to summarise a very large and diverse data sets, and in assisting the process of model selection.

During the analysis of the CCC metrics which are highly skewed and dispersed over a large range, the use of logarithmic transformation was necessary.

Complementary Cumulative Distribution Function

In studying the tail of a distribution, it is more convenient to plot the probability with which a variable that is grater than or equal to some value appears. The complementary cumulative distribution function (CCDF) plot, defined in the following equation, plots the probability $D_{comp}(x)$ of observing values greater that x.

$$D_{comp}(x) = P(X > x) = 1 - D(x) \tag{4.1}$$

When plotted in log-log axis, the linearity of the complementary CDF plot indicates the presence of a long-tail behaviour. A sample plot of a Gamma, Weibull and Pareto probability distributions is given in Figure 4.1. Clearly, the only linear function is the Pareto one, confirming the presence of a long-tail.

4.2.3 Measures of Location and Dispersion

One of the first tasks in describing a sample population is to measure its central tendency (or location on the number line), and estimate its dispersion (or how spread the values are on the number line). Even if measurements of a process with well defined statistical sample distributions are taken, some outlier data

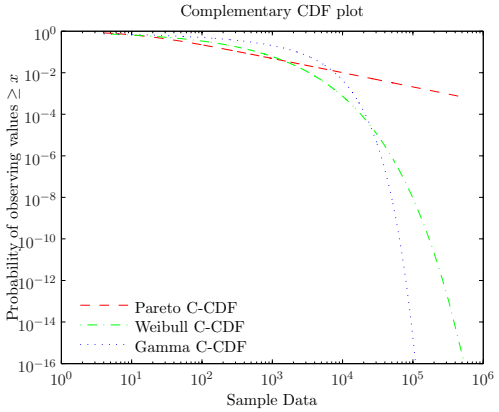


Figure 4.1: Sample complementary cumulative distribution plot (CCDF) of a Gamma, Weibull and Pareto distributions. The linearity of the plot indicates strong long-tail behaviour.

values are likely to occur. In the case of an empirical data set produced by sampling a process that has not been fully understood or statistically characterised, establishing its location and dispersion becomes challenging.

Intuitively, one would expect metrics collected from the CCC to have a very wide distribution of values. Reporting common single value statistics of those distributions is unlikely to offer much insight, could possibly confuse the reader, or misrepresent the real features of the data, but may still need to be reported for comparison purposes with other historical usage traces.

Mean and Standard Deviation

Both the mean and the standard deviation are susceptible to, and highly influenced by outlier values. They are most useful when considering samples with a normal probability distribution, or one that can be approximated by it.

Median, Inter-quartile Mean and Inter-quartile Range

For a probability function P, a median m satisfies the following inequality:

$$P(X \leq m) \geq 1/2 \leq P(X \geq m) \tag{4.2}$$

Medians will not change significantly in a presence of a small number of outliers, thus making it a more robust measure of the central tendency than the mean. Medians are primarily used for skewed distributions, as some of the workload distributions are anticipated to be.

To handle a large number of outliers, an inter-quartile mean can be taken by discarding the lowest 25% and the highest 25% of values and calculating the mean of the remaining samples according to the following equation:

$$\bar{x}_{IQM} = \frac{2}{n} \sum_{i=(n/4)+1}^{3n/4} x_i \tag{4.3}$$

When the median is used to report on the location of the distribution, the inter-quartile range is often used to describe its dispersion. It is calculated as a difference between the third and the first quartiles of a distribution, and is robust to outliers.

$$r_{IQR} = Q_3 - Q_1 \tag{4.4}$$

As an aid in visualising the central tendency and the dispersion of a sample population described using the median value and the inter-quartile range, boxplots similar to the one shown in Figure 4.2 will extensively be used. Introduced in the 1980s by John Tukey [169], they graphically depict the robust measures of variance (the box top and bottom edges represent the upper and lower quartile), and location (the red line in each box is the median value of the sample).

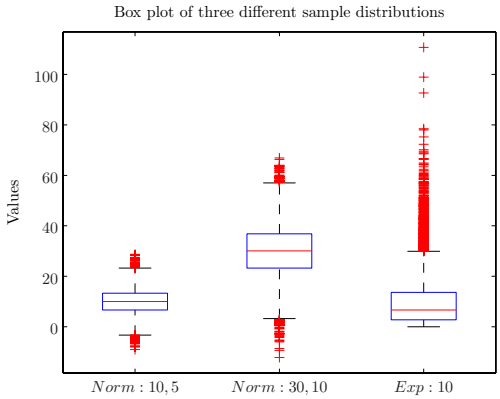


Figure 4.2: A sample Boxplot showing the central tendency (location) using median and dispersion using upper and lower quartiles and outlier values of an exponential and two normal distributions.

Sample values which are more than one and a half times the inter-quartile range away from the top or the bottom quartile are, by agreed notation, considered as outliers. The boxplot "whiskers" connect the highest and the lowest non-outlier values, while the red crosses are shown for each such outlier in the sample population. This definition of the outlier values applies throughout this thesis.

The sample graph shown in Figure 4.2 plots a boxplot for: (a) normal distribution with $\mu = 10$, $\sigma = 5$ (b) normal distribution with $\mu = 30$ and $\sigma = 10$ and (c) an exponential distribution with $e = 10$.

Coefficient of Variation

Defined as a ratio of the standard deviation and the mean, the coefficient of variation (CV) is used as a measure of the dispersion of a probability distribution:

$$\mathbb{CV} = \frac{\sigma}{\mu} \tag{4.5}$$

Distributions with CV$<$ 1 are considered of low variance, while those with CV$>$ 1 are considered of high variance. The coefficient of variation is mostly frequently calculated for the distributions whose standard deviations are significantly smaller then the mean. The violation of this assumption for many empirical distributions, and the CV's sensitivity to the changes in the standard deviation when the mean value is close to zero limits its usefulness. Nevertheless, it will be reported to facilitate comparison with other workload characterisation studies that have made extensive use of this metric [26, 173, 28].

4.2.4 Cyclic Behaviour

The existence of seasonal variations or cyclic behaviour is an important consideration in the time series analysis. The presence of such features indicates that the underlying process is not purely random, that certain correlation exists between

the time domain and the metric being analysed, and that the predictability of that metric would be increased if the relationship could be established.

The cyclic behavior in the computational workload was considered before, and was most notably modelled using Markov chains, for example by Song [155] and Thomas [156]. However, these studies were done on the traces of smaller, more dedicated and more specialised compute platforms. It would be of great interest to confirm such cyclic patterns exist on a large scale, multi-purpose production Grid.

The analysis of cyclic behaviour was considered with respect to variations of the observed metrics on the yearly, monthly, weekly and daily level. These seasonal periods were selected based on the assumption that the underlying workload is human submitted, research computing work. Graphical representation of the result was used throughout to aid in visualising the presence (or lack) of the cyclic patterns.

4.2.5 Scale Invariance and Self-similarity

Self-similarity, and the closely related concept of scale invariance, are properties of an object, function or a curve whose parts are similar to its whole. In other words, a self-similar curve or function looks "the same" when viewed at different scales. Mandelbrot, with his early work on fractals [174], introduced the notion of self-similarity which was later found in other processes, most notably local and wide area network traffic [148, 145], and the distribution of computer file sizes [141, 142, 144].

The concept of self-similarity is closely related to the long-range dependence and the power law relationships. A random variable X is said to have a *heavy-tailed distribution* if it satisfies the following equation [175]:

$$\mathbb{P}[X > x] \sim Cx^{-\alpha} \tag{4.6}$$

for some $C > 0$ and some $\alpha \in (0, 2)$.

The time series $\{X_1, X_2, \dots\}$ is said to be *weakly-stationary* if it has a constant and finite mean ($\mathrm{E}[X_i] = \mu$ for all i, where E means expectation) and the covariance between X_i and X_j (ie $\mathrm{E}[(X_i - \mu)(X_j - \mu)]$) depends only on $|j - i|$. For such time series, the autocorrelation function (ACF) $\rho(k)$ is given by:

$$\rho(k) = \frac{\mathrm{E}[(X_t - \mu)(X_{t+k} - \mu)]}{\sigma^2} \tag{4.7}$$

This definition allows the definition of *long-range dependence* [176] if the sum:

$$\sum_{k=1}^{\infty} \rho(k) \tag{4.8}$$

diverges.

The commonly used measure of long-range dependency and self-similarity is the Hurst parameter, which makes the assumption that the autocorrelation function follows the following, specific functional form:

$$\rho(k) \sim C_\rho k^{-\alpha} = C_\rho k^{2-2H}, \tag{4.9}$$

where $C_\rho > 0$ and $\alpha \in (0, 1)$ and $H \in (0, 1)$ is the Hurst parameter.

For $H \geq 1/2$ the process is considered self-similar with higher H values indicating stronger level of long-range dependence. Further discussion on the connections between the self-similar and long-range dependent process can be found in [177].

Due to its nature, the Hurst parameter is estimated, rather than calculated, using methods such as rescaled range (R/S) [178, 179, 175], variance analysis [175, 180] or wavelet spectral density approach [181, 182].

Using the rescaled range method [183], the Hurst parameter of an empirical series is estimated by calculating the average rescaled range over multiple regions of the data. For each region, the rescaled range is given by:

$$R(\tau) = \max[X(t, \tau)] - \min[X(t, \tau)] \text{ for } 1 \leq t \geq \tau \tag{4.10}$$

$$S(\tau) = \sqrt{\frac{1}{\tau} \sum_{t=1}^{\tau} [\xi(t) - \langle \xi \rangle_t]^2} \tag{4.11}$$

$$R/S = \frac{R(\tau)}{S(\tau)} \tag{4.12}$$

A linear regression line through a set of points, composed of $\log(n)$, where n is the size of the areas on which the average rescaled range is calculated, and the *log* of the average rescaled range over a set of regions of size n, is calculated. The slope of the regression line is the estimate of the Hurst exponent [174].

Figure 4.3 shows a sample plot estimating the self-similarity of samples drawn from a normal distribution using a rescaled range method. The regression line is of good fit, whose slope estimates the Hurst parameter at $H = 0.27$ which rightly suggest that this is not a self-similar and long-range dependent population.

When developing a predictive system to which a time series will be presented as an input, the effects of long-range dependency and self-similarity must be taken into account. The fact that the system will appear bursty no matter how aggregated it is requires a robust design that will not simply ignore or filter out the "spikes", but treat them as an intrinsic part of the process.

4.2.6 Metric Dependency and Correlations

The strength and direction of the linear relationship between two random variables is indicated by their correlation coefficient. It is generally accepted that

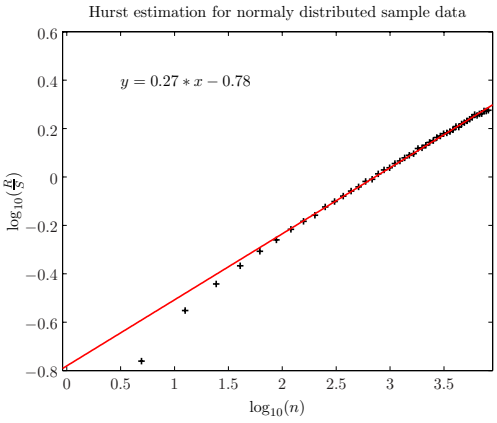


Figure 4.3: Sample plot of rescaled range (R/S) analysis used to estimate the self-similarity of samples and the Hurst parameter value. The normal distribution has the Hurst value of less than 0.50 an hence does not exhibit self-similarity.

correlation refers to the departure of the two variables from independence, and is commonly expressed in terms of their covariance:

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} \tag{4.13}$$

$$= \frac{\sigma_{XY}}{\sigma_X \sigma_Y} \tag{4.14}$$

where σ_{xy} is the covariance between the variables X and Y:

$$\sigma_{XY} = E[(X - \mu_X)(Y - \mu_Y)] \tag{4.15}$$

$$= E(XY) - \mu_X \mu_Y \tag{4.16}$$

and E is the expected value of the variable.

The correlation coefficient effectively scales the covariance by the standard deviation of each variable, and is thus a dimensionless quantity that describes the linear relationship between a pair of variables of different units. Crucially, the parametric correlation methods, such as the often used Pearson's product-moment coefficient [125], rely on the distribution means and standard deviations and the assumption of the normality of the sample distribution, and are less useful if such assumptions are violated.

Non-paramteric correlation coefficients, such as the Spearman's ρ and Kendall's τ [125], assesses how well an arbitrary monotonic function could describe the relationship between two variables, without making any assumptions about the frequency distribution of the variables. Spearman's rank correlation coefficient will be reported for applicable workload metric correlations, and is defined by the following equation:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{4.17}$$

where d_i is the difference between each rank of corresponding values of the two variables, and n is the number of pairs of values.

Random variables can often have non-linear correlations, such as in seasonal variation patterns or daily peak periods. The correlation coefficient is unable to detect these relationships, and a more general approach using correlation ratios is then warranted. This method is able to detect almost any functional dependency between random variables by comparing the statistical dispersion within individual categories to the dispersion across the whole sample population. If a reduction in dispersion is observed, the variables are correlated [184].

This approach will be used extensively in establishing the relationship between the job meta-data and its wallclock execution time. Sample dispersion metric (usually coefficient of variation) for jobs grouped by certain meta-data will be compared to the overall trace dispersion and reported using bar charts.

4.2.7 Locality of Sampling

The purpose of the majority of research work in the area of the Grid workload characterisation was that of generative modeling - trying to model the workload so that new, representative, workloads can be generated for Grid middleware testing. The traces used were of varying lengths, from a few days to several months. These periods are not sufficient to capture the high degree of workload variability both within a certain time period, and between different periods of time.

Correspondingly, the effect of large variations in the Grid workload observed over longer time scales was mostly neglected by the previous research in the this area. While this may be acceptable in terms of the generative trace modelling, from the aspect of the predictive scheduling, high variance of the job execution time and other relevant metrics poses a big challenge.

In this thesis, a novel approach in reducing this variance will be considered. By using a specially constructed plot, the variability of the important metrics will be compared on a sampling scale considerably smaller than the whole trace. The rationale behind this is that the workload is evolving in epochs characterised by larger variance between them and a more deterministic behaviour within each one.

An example of the plots used to study this trace feature is shown in Figure 4.4. It represents values (given by the colour intensity of each patch) of ten periodic observations (x axis) of ten sample variables (y axis). By reading the plot column by column (keeping x value constant and observing the difference along the y axis), the intra-period variations between the variable values can easily be seen. Equally clear are the variations of one variable between different

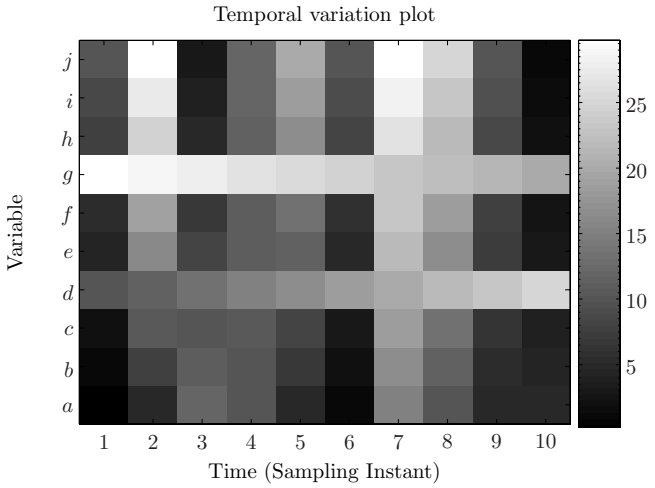


Figure 4.4: An example of a temporal variance plot showing the sample value fluctuation, shown as colour intensity, over short and long time scale.

time periods observed by reading the plot row by row (keeping y value constant and observing the difference along the x axis).

Characterising the CCC workload, the variance was so large it was often necessary to colour the patches by using the natural logarithm of the observed value. Nevertheless, these plots are very valuable in understanding the level of fluctuations both within and between workload epochs, and one of the motivations for including the temporal job properties into the forecasting models detailed in Chapter 5.

4.3 General Workload Properties

The workload analysis was done on the trace spanning the twelve months of 2005 and compromising more than six hundred thousand jobs. During this period, a total of 37 users were active, and have submitted a varied and highly dynamic workload.

Considering the length of the workload trace, large number of data points, and the complexity and interdependency of metrics, the analysis will begin by introducing general properties of the workload. The purpose of this section is to:

- Present the important workload metrics using the run-sequence and cumulative distribution function plots.
- Investigate the presence of cyclic behaviour.
- Establish the statistical properties of the metrics, including normality, long-tailedness and self-similarity

Four primary metrics will be discussed: the arrival rate and the inter-arrival time, queue time, wallclock execution time, and memory utilisation. For each, a

run-sequence plot will give an overall picture while the CDF graph will show the distribution of the observed values and indicate candidate model distributions. Probability plots will test the normality of the value distribution and the complementary CDF plots will assess the length of the distribution tail and its fit to one of the frequently used distributions. Finally, a rescaled range analysis will be used to estimate the Hurst parameter and the degree of self-similarity of the data.

4.3.1 Workload Summary

The summary of the CCC trace is given in Table 4.3.

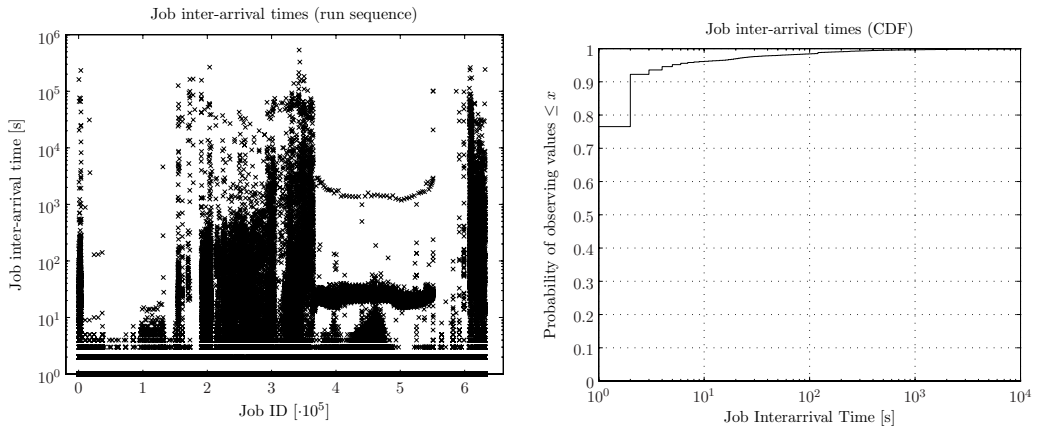
First job time	01.01.2005 13:45
Last job time	20.12.2005 12:28
Number of days	353
Number of recorded jobs	646,045
Number of valid jobs	632,027
Unique users	37
Unique Virtual Organisations	27
Unique job names	2,268
Total job wallclock time	2,721,157,784s (31,495 days)
Total job CPU time	2,212,915,331s (25,612 days)
Mean Cluster Utilisation	89%
Mean Application Efficiency	81%
Deleted (missing jobs)	5,792
Failed (0 sec) jobs	15,625

Table 4.3: The summary of the CCC workload analysed

The quality of the accounting file was acceptable, with about 2% of invalid entries (missing or corrupted fields). By comparing the range of unique Job_IDs and the total number of recorded jobs, it was found that less than 1% are missing. The cause of this could be the removal of jobs from the queue before they were executed, or some other systematical problem with the accounting system.

Another 2.5% of the jobs have executed for less than one second, the sampling accuracy of the accounting file. While it is possible that these jobs were meant to run for such short time, it is not likely that the users would submit such short jobs to a Grid facility. They are therefore considered as failed, most probably due to an error in the initial setup of the executable environment. This failure rate is considerably less than previously reported by Cirne [23] or Li [26] for example.

Overall cluster utilisation during the period in question was 89%, calculated as a ratio of the real time and the total used wallclock time multiplied by the number of worker nodes, was higher then anticipated or previously observed on other Grid clusters [23, 26]. Mean application efficiency, the ratio between the wallclock time and the CPU time the job has used, was also very high indicating



(a) Run sequence plot (b) Distribution function

Figure 4.5: Job inter-arrival times: (a) run sequence plot and (b) distribution function showing 75% of job inter-arrival times are less than one second.

that the applications submitted to the CCC were highly optimised, and that the workload was predominately compute bound.

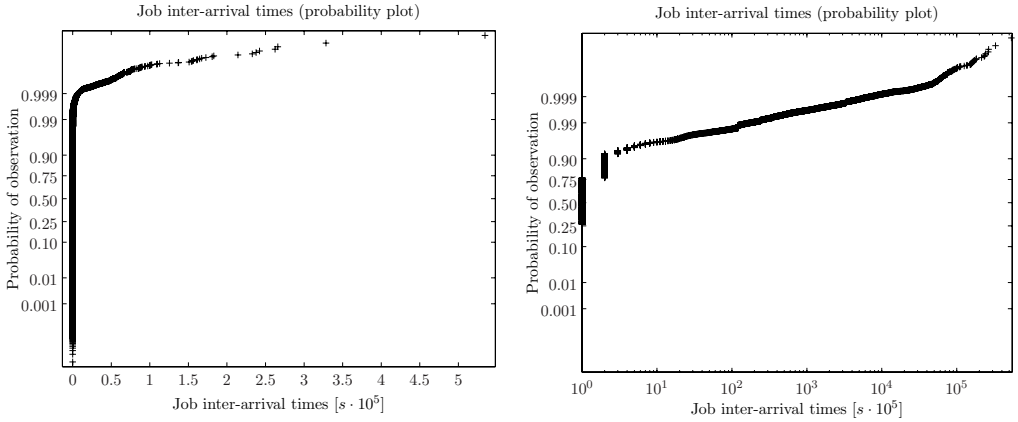
4.3.2 Arrival Process

The job inter-arrival time is defined as a difference between the submission times (Sub_Time) of two consecutive jobs. Since these times are recorded as UNIX epoch times, the resolution of the measurement is one second. Figure 4.5 shows the run-sequence plot of the job inter-arrival times for the whole year, and distribution of values in a CDF plot.

The arrival pattern is clearly very bursty: more than 75% of jobs arrive less than one second apart, and less than 1% of jobs arrive more than three minutes apart. Considering that the cluster was open for job submissions continuously, it is not unreasonable to expect a steady stream of jobs arriving throughout the year. The dynamics of this process will be discussed in more detail in Section 4.6.

The implication of this arrival pattern on the scheduling process is that the jobs are very likely to be submitted in large batches, followed by a "quiet" period. As it will be shown later, the peak and off-peak submission periods can, to a great extent, be forecasted and scheduling actions taken to brace for the high volumes of job submissions.

A normal probability plot was constructed in order to test the normality of the job inter-arrival distribution. Figure 4.6(a) shows a significant skew towards smaller values of the inter-arrival times. This plot is clearly non-linear, and the assumption of normality cannot be made. The second plot, Figure 4.6(b), shows the normality of logarithmically transformed job inter-arrival times. Apart from the highly probable values between zero and three seconds, the remainder of the



(a) Linear scale (b) Logarithmic scale

Figure 4.6: Job inter-arrival times: normal probability plots in (a) linear and (b) logarithmic scale. Apart from the evident skew between 0 and 3 seconds, the plots indicate inter-arrival times are otherwise log-normal.

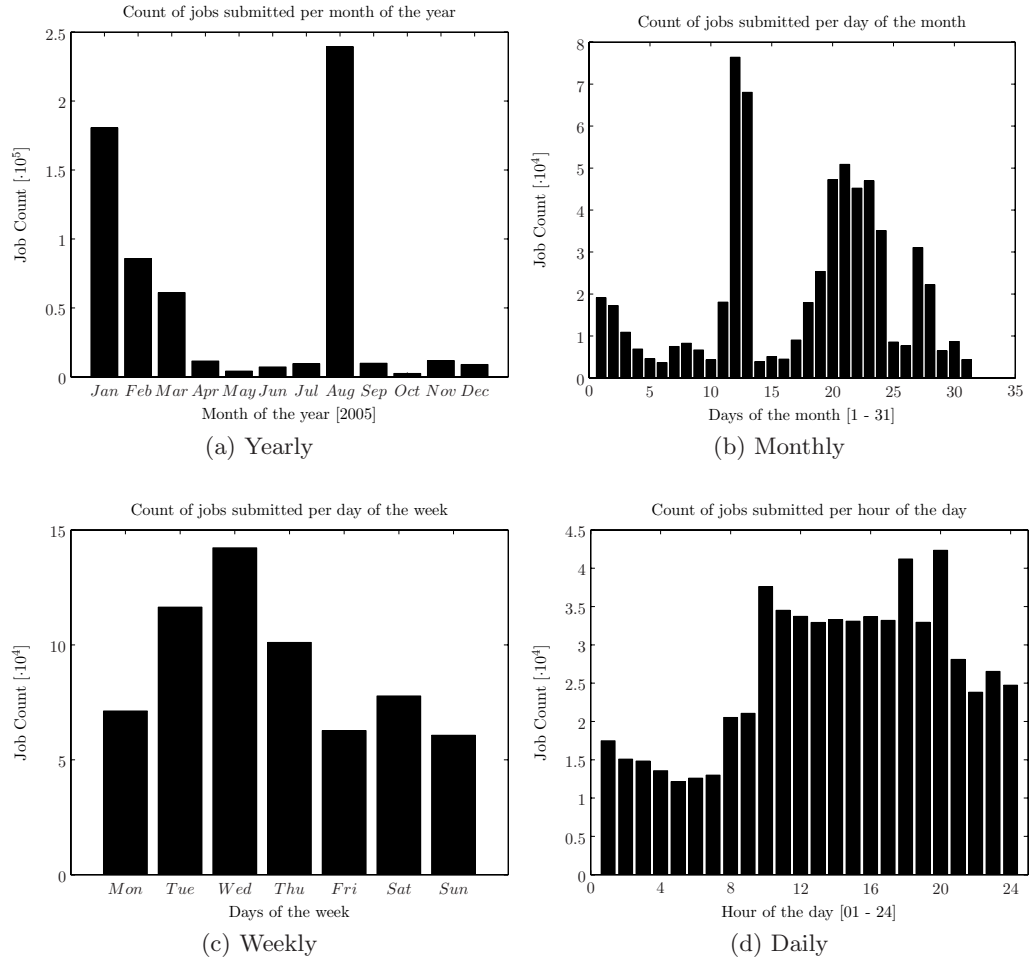
plot shows very good linearity. The job inter-arrival times are thus log-normal for values greater than 3 seconds.

The cyclic behaviour of the total number of submitted jobs is plotted in Figure 4.7. The observable daily cycle is representative of the usual human work flow: job submissions increase at the beginning of the day (8am) then dip slightly around lunch hour (1pm), followed by another strong peak at the end of the work day (6pm to 8pm), and a steady fall off during the evening and night hours. Such fluctuation indicates a user tendency to submit jobs as they arrive to work and just before they leave, anticipating their execution overnight. A sharp rise in job submissions between 8am and 10am, and a more gradual fall-off in late evening and night hours can indicate different work practices among users (some people prefer to work until late, but most come in until 10am).

The weekly pattern shows almost anecdotal features with a steady rise in job submissions from Monday to Wednesday followed by a decrease until Friday. Both weekend days show a significant number of job submissions, with Saturdays comparable with Fridays and Sundays with Mondays. Here, users may be unintentionally load balancing the system, anticipating better turnaround times for the jobs submitted in what they perceive as the off-peak periods. The weekend submission count is certainly further increased by the ability to log into the CCC facility remotely, although this could not be fully established from the data collected.

The monthly cycle seems to be dominated by the weekly pattern with a strong peak at around the middle of the month. Fluctuations between different months of the year 2005, and a sudden jump of job submission in August are most probably influenced by the research timetables of the CCC users.

The shape of the tail of the inter-arrival times distribution, and its fit to



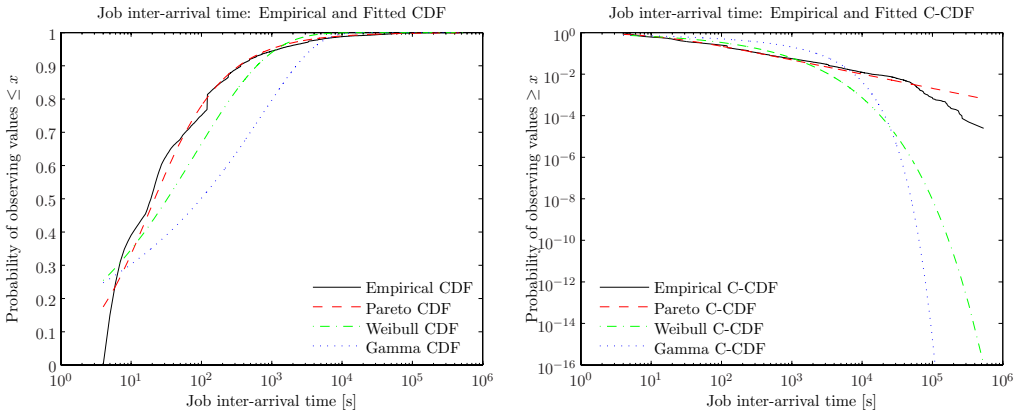
(a) Yearly (b) Monthly

(c) Weekly (d) Daily

Figure 4.7: Job submission: the number of jobs submitted in each time period within 2005 - (a) yearly cycle is not representative as only one year's data has been collected, (b) monthly cycle shows tendency to submit more jobs toward the end of the month, (c) weekly cycle shows mid-week surge and weekend dip, while (d) daily cycle shows strong human working patter with 8am-8pm peak.

some commonly used distributions is given in Figure 4.8. Shown are the inter-arrival times greater than 3 seconds (approx. 7% of all values). The Pareto distribution gives the best fit to the empirical data which contains few very large values (largest one 534511 or more than 6 days). These extreme values are most probably caused by a failure of the external network connectivity or the cluster downtime, but are nevertheless a reality in a production environment.

Rescaled range analysis of the job inter-arrival times is shown in Figure 4.9. The plot shows good linearity, with the Hurst exponents estimated at $H = 0.85$. This high level of self-similar behaviour indicated that the arrival process is bursty on all time scales. Certainly one of the main reasons for such behaviour is the on/off pattern of the job submissions, and a very skewed, long-tailed distribution of the job inter-arrival times.



(a) CDF (b) Complementary CDF

Figure 4.8: Job inter-arrival times ≥ 3 seconds: CDF and complementary CDF are used to judge the presence of long-tail behaviour and estimate the best fitting model. Pareto function describes the empirical data well over more than five orders of magnitude.

4.3.3 Queue Wait Time

Queue wait time is the delay the job experiences from its submission into the Grid to the actual start of the execution on one of the worker nodes. Assuming sequential jobs which are being executed in a FIFO order, the job queue time is the sum of the wallclock execution times of all jobs preceding it in the queue. Job queue wait times are hence directly related to the job submission process and the job execution times.

The plot in Figure 4.10 shows the queue wait times for each job submitted to the CCC cluster, and the corresponding cumulative distribution function. The values have been derived from the trace by subtracting the recorded job start time (Start_Time) from the job submission time (Sub_Time). The resolution of the measurements is one second.

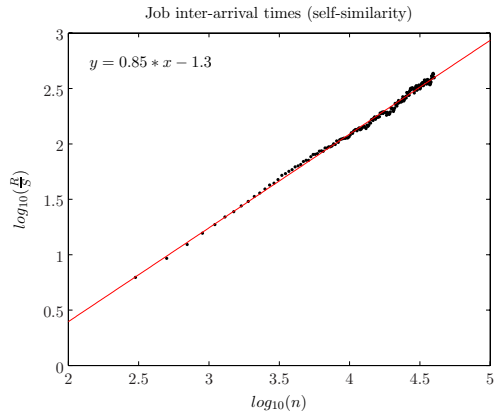
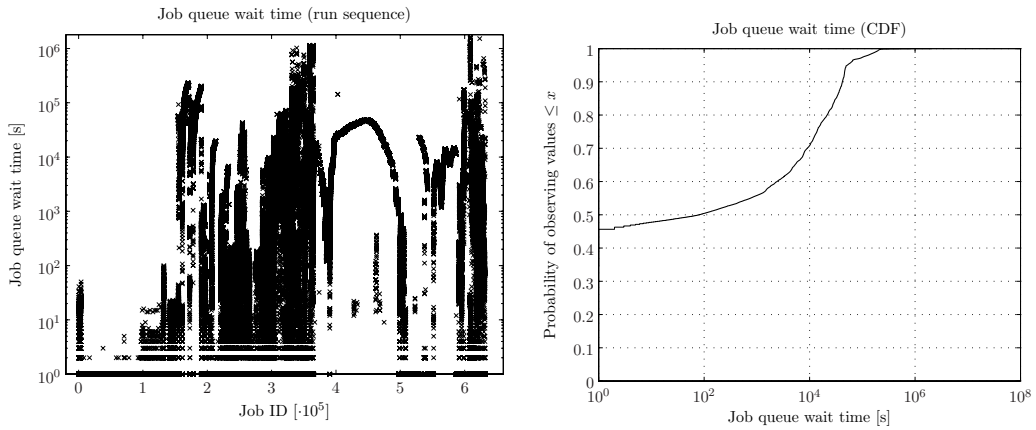


Figure 4.9: Job inter-arrival times: Hurst parameter, as the measure of self-similarity, was estimated using rescaled range (R/S) method to $H = 0.85$.

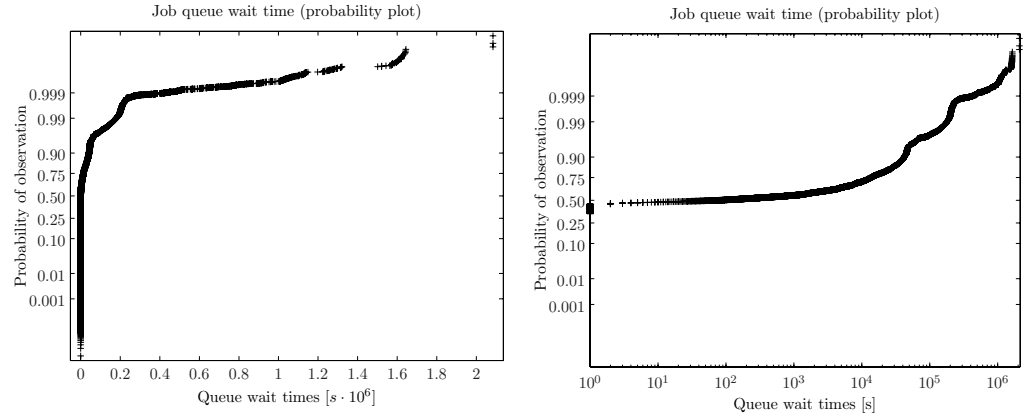


(a) Run sequence plot (b) Distribution function

Figure 4.10: Job queueing times: (a) run sequence plot and (b) distribution function revealing that 45% of submitted jobs execute immediately and without any queueing delay.

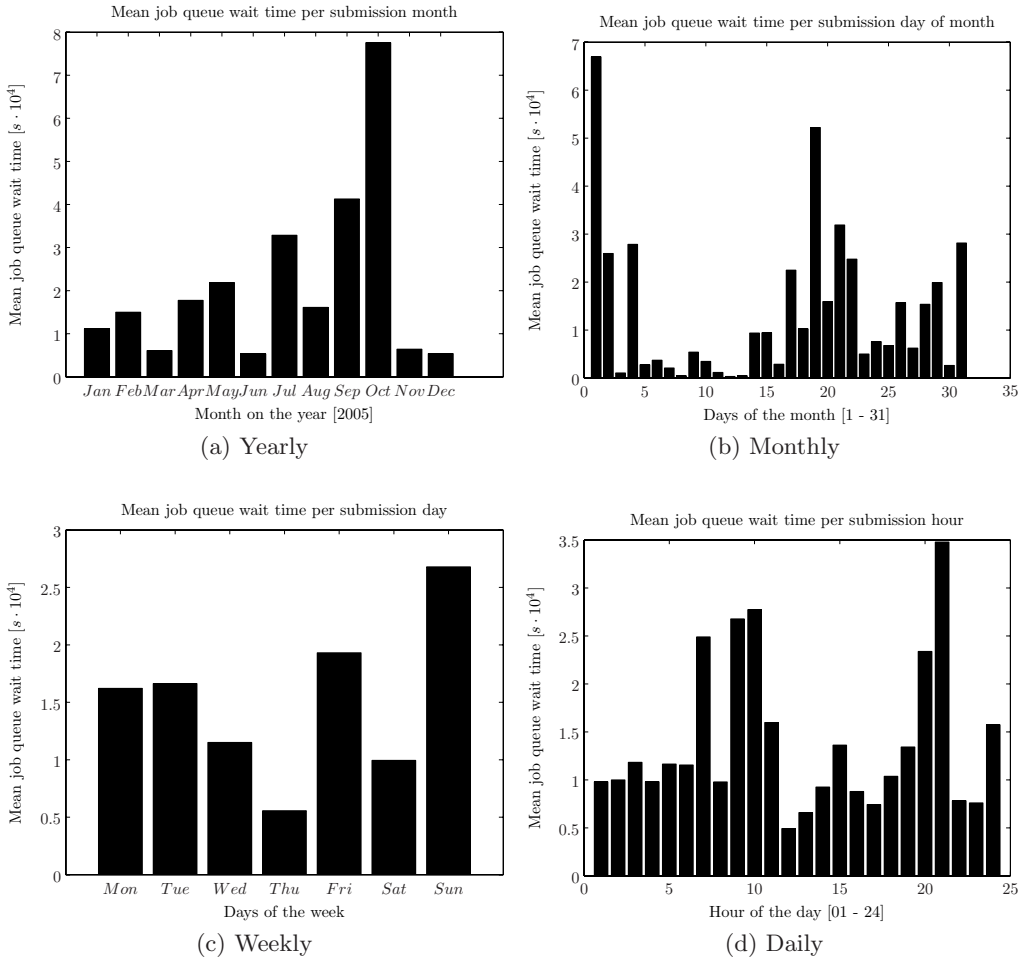
Despite the high level of overall cluster utilisation, the measurements indicate that approximately 45% of the jobs have been started as soon as they were submitted (queue wait time of less than one second), and approximately 95% have begun executing less than 12 hours from the submission. However, some very long queue wait times have been observed, and can not be attributed to the scheduled system down time, as queues have been purged in advance of these events.

The normality of the job queue wait times is significantly influenced by the already mentioned high proportion of jobs starting their execution immediately. The probability plot, shown in Figure 4.11(a), exhibits very poor linearity up to the queueing time of $2 \cdot 10^5$, and only moderate linear behaviour afterwards. The



(a) Linear (b) Logarithmic

Figure 4.11: Job queueing times: normal probability plots in (a) linear and (b) logarithmic scale. Poor linearity in both plots indicates queuing times distribution could not be considered neither normal nor log-normal.



(a) Yearly (b) Monthly

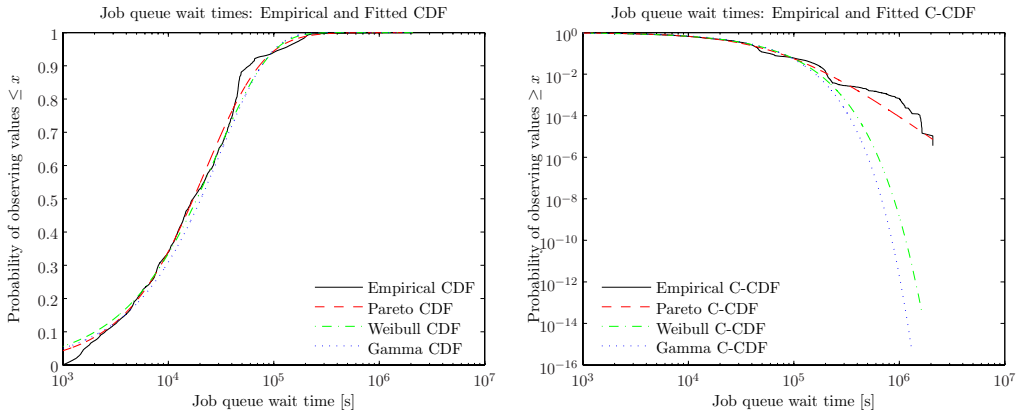
(c) Weekly (d) Daily

Figure 4.12: Job queueing times: the average amount of time a job has queued based on its submission time on (a) yearly, (b) monthly, (c) weekly and (d) daily level. The plots reveal positive correlation between job submission process and queueing time.

queueing times are not log-normal either, as demonstrated by the plot in Figure 4.11(b).

Figure 4.12 shows the variation of queue wait times a job experiences depending on the time of its submission. Again, the daily cycle is strongly influenced by the user work habits, and directly complements the job submission count cycle plot given previously (Figure 4.7). Jobs submitted at morning and evening peak hours experience significantly longer queuing times then those submitted at other times of the day. Interestingly, jobs submitted at lunch hour have the shortest waiting time, despite being preceded by a large number of morning job submissions.

The weekly cycle may seem at odds with the job submission cycle, since the day with the most job submissions (Wednesday) has one of the lowest queue wait times, while Sunday has the largest. However, the job queue wait time is dependant on the number of jobs already queueing and the sum of their execution



(a) CDF (b) Complementary CDF

Figure 4.13: Job queuing times ≥ 1000 seconds: CDF and complementary CDF are used to judge the presence of long-tail behaviour and estimate the best fitting model. Pareto function provides the best fit to the empirical data.

time which produce a lag between the peak of job submission and the peak of queueing times.

This effect is clearly seen in the yearly plot, where a large number of jobs submitted in January and August (shown in Figure 4.7) lead to a gradual increase in the queue wait times up to two months later. The monthly plot of the queue wait time cycle is again of little value, its features dominated by the weekly cycle and showing no other clear seasonal effects.

Plots of the tail of the queue wait time distribution are given in Figure 4.13. For the queue wait time values greater than 1000 seconds, the Pareto distribution provides the closest fit, and the linearity of the complementary CDF of empirical distribution indicates the presence of the long-tails.

The rescaled range method for estimating the self-similarity of the job queue wait time, Figure 4.14, estimates the Hurst exponent value at 1. This is the highest theoretically possible value, and while the method is not an exact calculation, it certainly indicates an extremely mean-averting and self-similar process. But since the job queue wait times are a function of the arrival process and the job wallclock execution times, both of which are heavily self-similar themselves, such result is not unexpected.

4.3.4 Wallclock Execution Time

From the scheduling aspect, the wallclock execution time is the most important metric, and one from which queue wait time and the job makespan* can be calculated.

*Time taken from the job submission to the job completion, usually equals queue wait time plus the job wallclock execution time

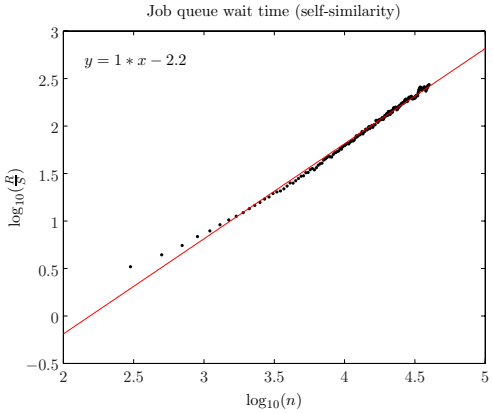
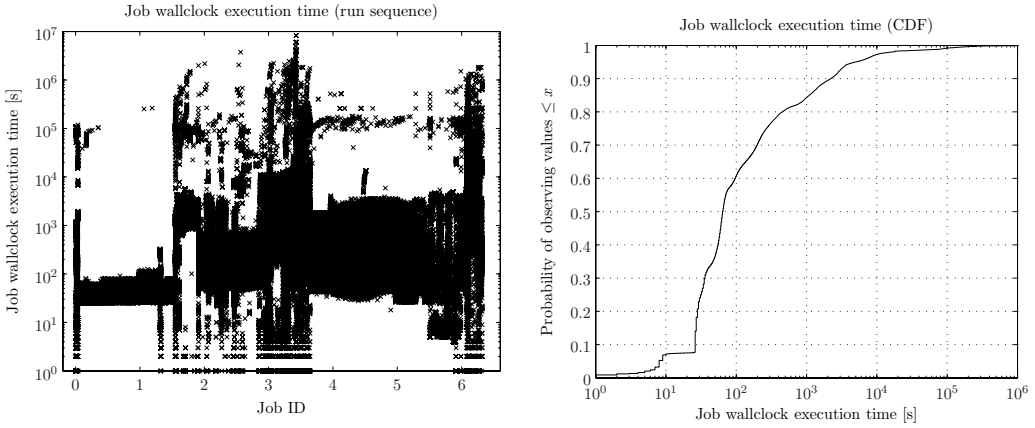


Figure 4.14: Job queueing times: The Hurst parameter, as the measure of self-similarity, was estimated using rescaled range (R/S) method to $H = 1$. This highest theoretically possible value indicates a very strongly self-similar process due to its dependence of job arrivals and runtimes, both of which are strongly self-similar.

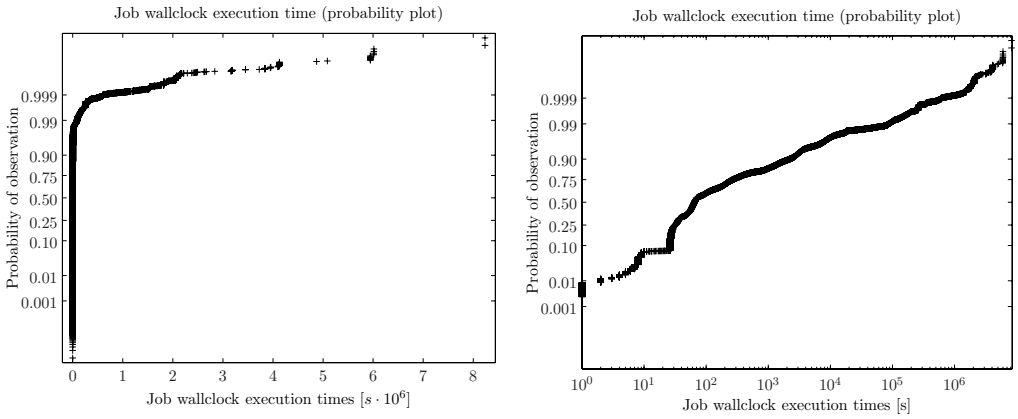
The run-sequence plot of the job wallclock execution times, and their cumulative distribution function are given in Figure 4.15. The run-sequence plot reveals a very large range of job execution times, from one second to more than three months, periods of relatively low activity and periods of high execution time variability. The features of the CDF plot indicate a low occurrence of jobs taking less than 25 seconds (around 0.07%), and an equally low number of jobs taking more than about a day to run (approximately 1% of jobs run for more than 10^5 seconds).

Such a distribution of the job execution times is likely caused by the user's selection of the jobs they are to submit to the Grid facility. As submitting each



(a) Run sequence plot (b) Distribution function

Figure 4.15: Job wallclock execution times: (a) run sequence plot and (b) distribution function demonstrating that apart from a small number of very short or very long jobs, each runtime is as likely as any other.



(a) Linear scale (b) Logarithmic scale

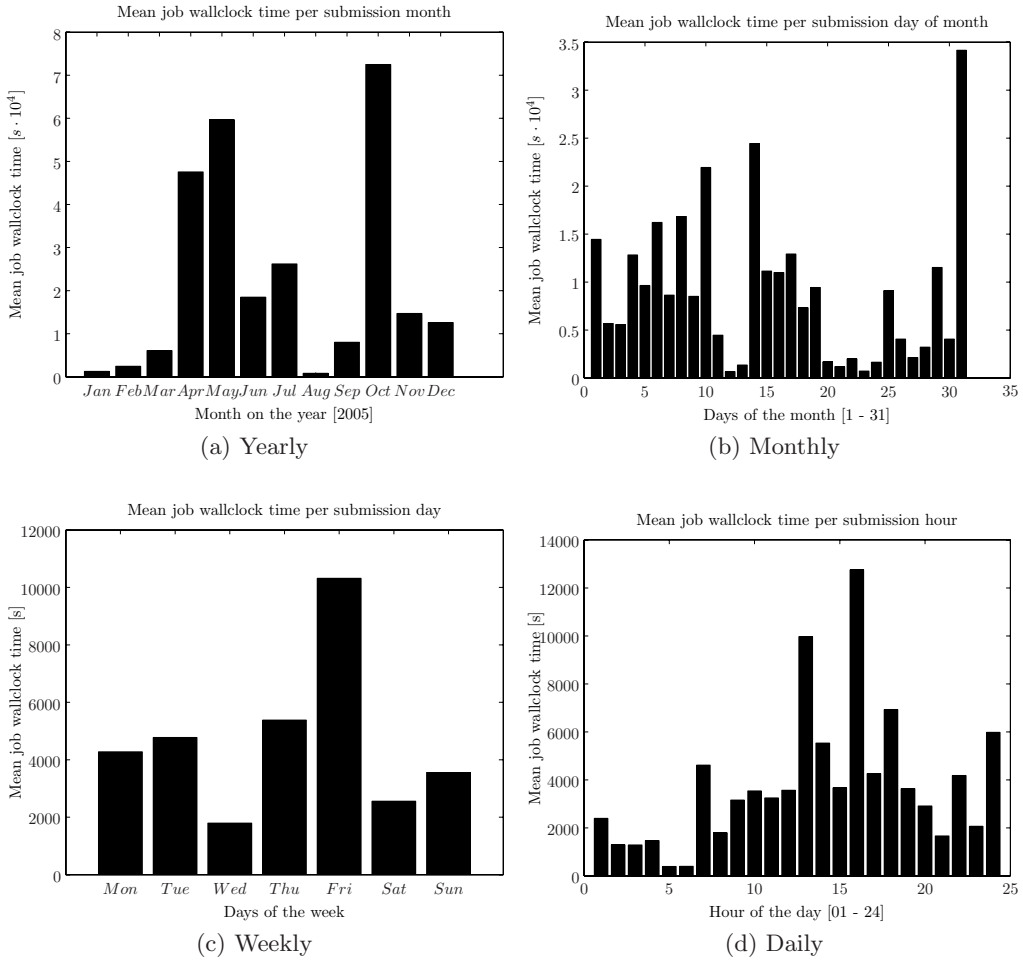
Figure 4.16: Job wallclock execution times: normal probability plots in (a) linear and (b) logarithmic scale. The latter demonstrates good linearity with significant departure only at the low end supporting the suggested log-normality of job runtimes.

job to the CCC presents an administrative overhead to the user, they are likely to choose to run shorter jobs on their local workstations. Equally, as most users are to some extent aware of the performance of their applications, and the hardware on which it will be run on the CCC, they are unlikely to submit regular jobs which will take an amount of time much larger then what a normal human workflow would consider acceptable (for example a day or a weekend). Understandably, in some circumstances users would have no other options and would rather wait a very long time for a job to complete then not to run it at all.

The remainder of the execution times form a continuous distribution with no steps or observable modes, indicating that every execution time from $2 \cdot 10^1$ to 10^4 is almost as likely to occur as any other from the same range. The Grid resource management and scheduling systems should be developed in accordance with such expected load, avoiding the assumption of any "preferred" values of the job execution times.

The normal probability plot of job wallclock execution times is shown in Figure 4.16. The normality can certainly be assumed on the linear scale of values, as very strong skew exists towards smaller values. However, logarithmically transformed values do show a very strong linear tendency throughout the whole range, with some significant departures only at the very short running jobs. This property of job execution times has been noted by other researcher analysing distributed machine traces [147, 25, 185], and can now be confirmed in the case of a multi-purpose production Grid as well.

Figure 4.17 shows the cyclic variation of the mean job wallclock execution times according to the time of their submission. Again, the daily variation shows strong peaks at the beginning, middle and the end of the work day. The most



(a) Yearly (b) Monthly

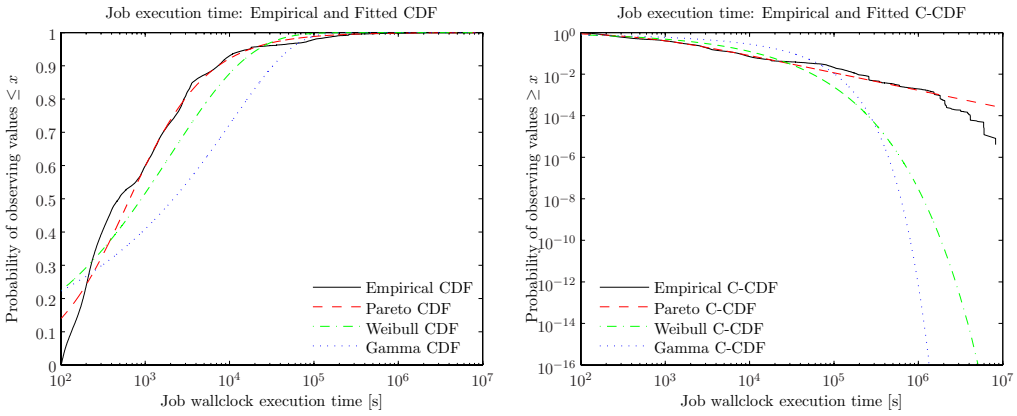
(c) Weekly (d) Daily

Figure 4.17: Job wallclock execution times: the average job runtime based on its submission time on (a) yearly, (b) monthly, (c) weekly and (d) daily level. Weekly and daily plots reveal strong tendency to submit longer running jobs on Fridays, mornings, just before lunchtime and at day's end.

prominent execution time peak at around 4pm ($\approx 3\frac{1}{2}$ hours) is almost 3 times larger than the mean job execution time at the beginning of the day ($\approx 1\frac{1}{4}$ hours). Intuitively or purposely, users rely on their limited insight into the complexity of their jobs to submit shorter ones for execution during their work day, leaving longer running jobs for overnight runs.

A very similar picture emerges from studying the weekly cycle. The shortest running jobs are submitted on Wednesday, the day with the highest count of job arrivals, while the jobs submitted on Fridays are by far the longest running ones. Again, users are trying to adapt the workload to their work cycle by running shorter, perhaps test or tuning, jobs during the week and longer ones over the weekend.

The yearly plot, to some extent, indicates the fluctuations during the academic year, but as it is based on only one year's worth of data, and as its scale is very long compared to most of the scientific tasks, it is only suitable for informational



(a) CDF (b) Complementary CDF

Figure 4.18: Job wallclock execution times $\geq$ 100 seconds: CDF and complementary CDF are used to judge the presence of long-tail behaviour and estimate the best fitting model. Pareto function provides the best fit, especially for values greater than 1000 seconds.

use. Looking at the month of August however, it is clear that a high arrival rate may not lead to high contention on the cluster. Again, monthly variations do not yield significant insight as they seem to be dominated by the weekly cycle.

The behaviour of the execution time tails is examined in Figure 4.18. Tail cut-off points of 100 seconds, compromising around 40% of the total number of jobs, has been used, with the Pareto, Weibull and Gamma distribution functions fitted to the empirical data. The Pareto distribution exhibits a very good fit over almost five orders of magnitude, with only a small overestimate of the probability of the longest running jobs ($\geq 2 \cdot 10^4$).

The estimation of the self-similar nature of the job wallclock execution times using the rescaled range methods is shown in Figure 4.19. A well fitting regres-

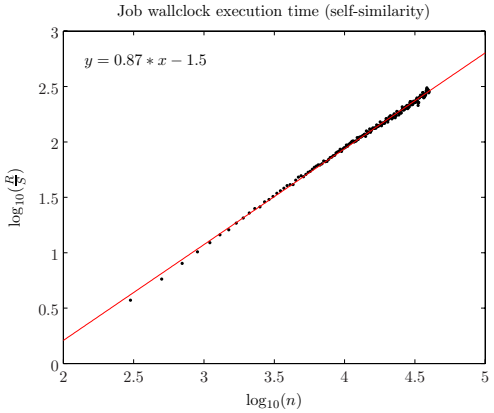
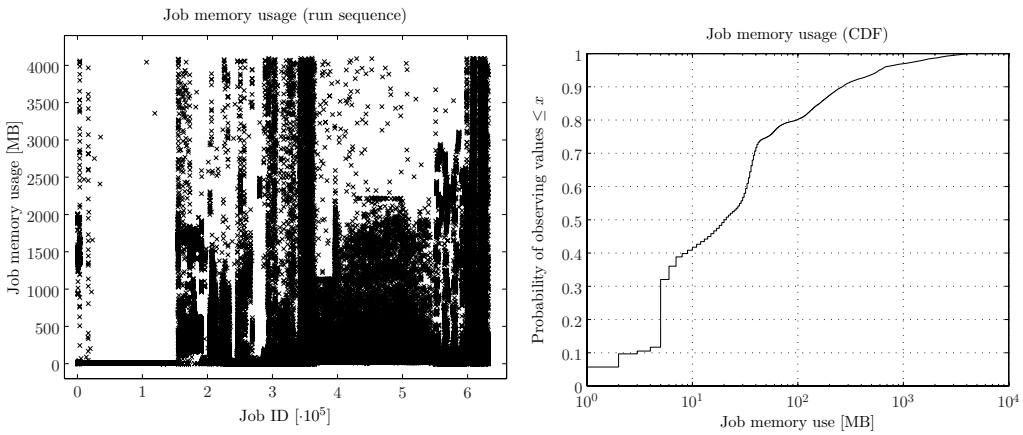


Figure 4.19: Job wallclock execution time: Hurst parameter, as the measure of self-similarity, was estimated using rescaled range (R/S) method to $H = 0.87$ indicating a strongly self-similar process.



(a) Run sequence plot (b) Distribution function

Figure 4.20: Job memory utilisation: (a) run sequence plot and (b) distribution function showing no obvious modality and a memory usage of less than 10MB by 40% of submitted jobs.

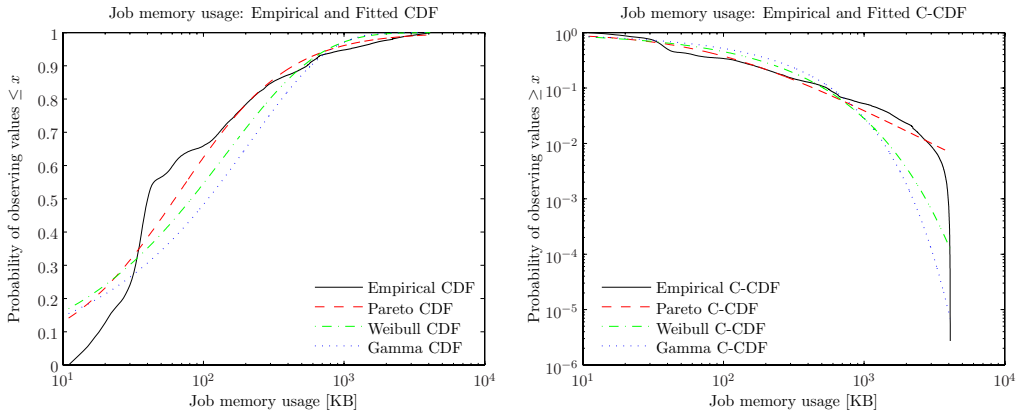
sion line estimates the Hurst exponent value of $H = 0.87$, indicative of a very scale invariant and self-similar process. Considered together with the previous analysis of the value distribution, the results confirm the strong non-linearity of the wallclock execution times and invalidate its approximation with the Poisson distribution used in the previous cluster scheduling research [152].

4.3.5 Memory Utilisation

Since memory allocation by an application is dynamic, a number of approaches can be taken in recording it. Memory use of a specific process can be recorded as a time series (such as in the Ganglia Monitoring System described in Section 3.4.1), or as a mean or maximum amount of memory allocated over a period of time. The value recorded by the Sun Grind Engine accounting file is the product of the job execution time and its average memory consumption yielding a metric in GBytes seconds. For the analysis presented here, this recorded value was divided by the job execution time to yield the average memory footprint of each application. The run-sequence plot and the distribution of values are give in Figure 4.20.

The time plot shows a significant variation of the memory use between jobs throughout the trace duration. The distribution function plot reveals that around 40% of jobs use less than 10 MBytes of host memory, after which the distribution continues in a log-normal fashion up to the maximum value of 4096 MBytes which is set by the physical amount of memory installed in the Grid nodes.

Contrary to some published analysis of the cluster job memory utilisation [186, 26], no prominent modality of the allocated memory has been observed. Previous work explained their existence by the frequent use of common shared libraries which require a fixed amount of memory, but without a more granular



(a) CDF (b) Complementary CDF

Figure 4.21: Job memory utilisation $\geq$ 100KB: CDF and complementary CDF are used to judge the presence of long-tail behaviour and estimate the best fitting model. None of the proposed functions offer an adequate fit over the whole range, although Pareto model does describe the general shape of memory utilisation distribution.

monitoring data this could not be established for the case of the CCC.

Figure 4.21 analyses the tails of the memory utilisation distribution with a cut-off point of 10 MBytes. While the Pareto function does describe the general shape of the tail, the fit is significantly poorer then for previous metrics, and the abrupt limit on the maximum value imposed by the hardware is obvious. Should modelling the memory use be of special interest, alternative distributions, or piece wise approximations using one of the distributions shown here should be used.

4.4 Workload Diversity

To this point, the workload was analysed as a monolithic set, treating each submitted job the same regardless of its associated properties (meta-data). While this approach gives an overview of the whole trace, it does not reveal the behaviour of its constituent parts, nor does it addresses the differences between them. As previously stated, one of the main premises of this work is the assumption that the highly variable and seemingly random behaviour of the whole workload is in fact a superposition of a number of different, and more predictable, patterns of job arrivals and execution times.

The purpose of this section is to analyse the job properties which are recorded in the accounting file and try to decompose the whole trace into a number of less variable, more predictive groups. It will demonstrate that in a general purpose, production Grid facility, a wide range of users submit jobs with widely varying resource requirements resulting in a highly dynamic workload. Modelling, or trying to predict, this compound load would therefore be much more difficult,

and less accurate, than partitioning it into smaller, and more consistent, clusters of similar "behaviour" and forecasting these constituting parts.

The previous section has already used the basic job meta-data such as the submission, start and end time stamps to show the cyclic nature of the user behaviour. In the following section the analysis of the other three job properties recorded in the accounting file will be used:

1. User - identifies the user whose credentials were used to submit the job.
2. VO - records the Grid Virtual Organisation to which the submitting user belongs.
3. Job name - contains the name of the job or application that has been submitted to the Grid.

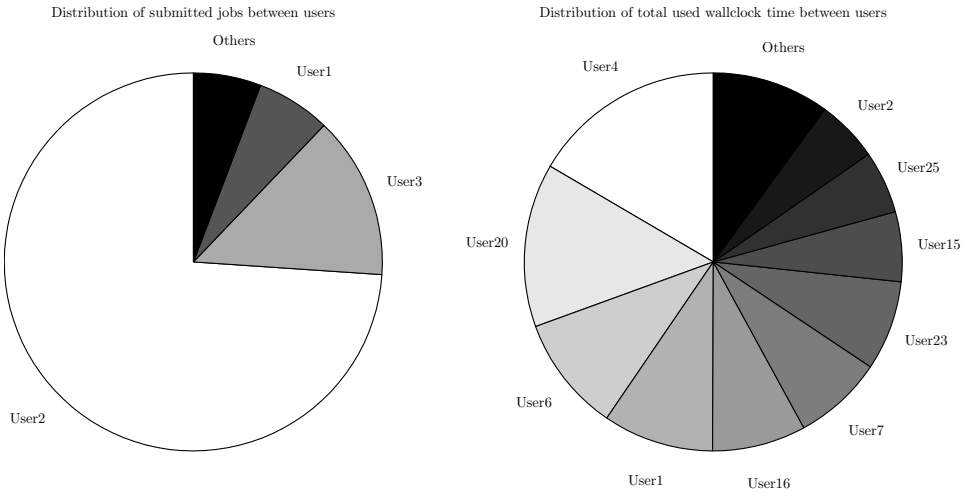
For each of these job properties, the aim was in establishing the following:

- The relationship between the above three items of the job meta-data, such as the number of users in each Grid VOs, or the number of different job names run by each user.
- The share of the total job count, or total wallclock time, for different users, VOs or job names.
- The level of the application efficiency for the jobs submitted by different users and VOs, or most frequently submitted job names.
- The distribution of the job wallclock execution times both between the users, VOs and job names, as well as within those categories.

4.4.1 User Differentiation

Considering a large number of different users of the CCC facility, the way in which each of them would use the facility was the first to be studied. The pie plot in Figure 4.22(a) shows a substantial domination of three users in the total number of jobs submitted to the system. Just *User2* accounts for almost 75% of all job submissions, and together the three most active users account for over 95% of all jobs submitted to the system. With such an imbalance, a question may arise whether some of the users are monopolising the CCC facility for their exclusive benefit.

The plot of the proportion of the total wallclock execution time used by the jobs belonging to the most active users, shown in Figure 4.22(b), shows a very different picture. It is clear that the users with the overwhelming number of job submissions tend to run very short jobs, and the total execution time is almost equally divided between the top 10 users. All other users amount for a significant proportion of the compute time as well.



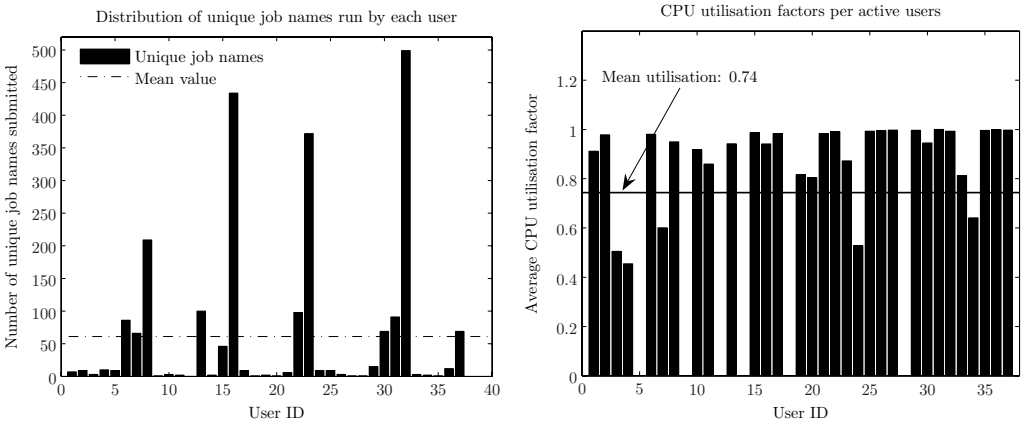
(a) Job count (b) Total wallclock runtime

Figure 4.22: User differentiation: by (a) submitted job count, and (b) total wallclock execution time reveal that few users submit a large proportion of jobs, while the distribution of total consumed compute time is more balanced. The high average CPU utilisation factor indicates jobs are mostly compute bound.

From the Figure 4.23(a), which plots the number of unique job names submitted by each user, it is clear that the distribution is modal and characterised by the majority of users mostly submitting jobs with the same name, ten or so users submitting between 50 and 100 different job names, and few users submitting jobs with several hundred different names. This wide gap is the testimony to the different workflow management between the users, with some preferring generic names while other tend to make their job names unique for each run. Introduction of a standardised workflow management system, able to uniquely identify different applications making up the workflow would in many ways alleviate these issues and enable much more insight into application behaviour.

Figure 4.23(b) shows that the CPU utilisation levels are very high, with the average at 74%, indicating that the majority of the submitted jobs are compute bound. If a few users that have submitted no jobs, and a few that had a very low CPU efficiency barely registering on the plot, were excluded, the actual average CPU utilisation would have been even higher. Considering that the workload almost exclusively consisted of sequential jobs, these results indicate that any data staging that was required was executed prior to the submission of the job into the Grid. This greatly reduces the effect that network performance has on the length of the job execution.

The distribution of the wallclock execution times also differs significantly both within the jobs submitted by a single user, and between different users. Figure 4.24 shows cumulative distribution function for the four most frequently run jobs by the user submitting the highest number of jobs (*User2*). Steep slopes of the



(a) Unique job names per user (b) CPU utilisation factor

Figure 4.23: User differentiation: by (a) unique job names count and (b) CPU utilisation factor. Users tend to either submit job with generic names or user a unique name for each job run. The very high CPU utilisation factor suggests most jobs are compute bound with the network performance having a limited influence on their execution times.

CDF plot for executables 7, 9 and 13 indicate a very narrow distribution of the job runtimes, with a small variance ideally suited for forecasting. Executable 38 also exhibits similar behaviour, but with certain modality and preference to the execution time of either less than 10 seconds, or between 30 and 100 seconds.

From the presented plots and analysis, it is clear that the user "behaviour", including the number, the type and the distribution of the runtimes of the jobs they submit, vary significantly between them. It has also been demonstrated that even further differentiation is possible by looking at the properties of the different job names a single user submits. This presents a valuable insight in the

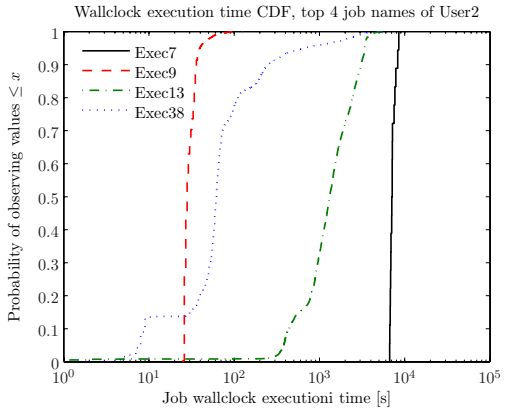
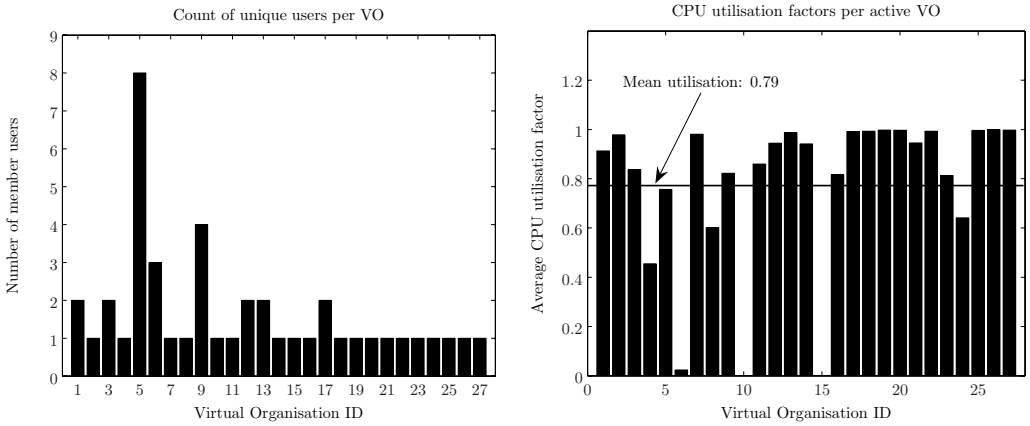


Figure 4.24: User differentiation: comparison of distribution functions of the job wallclock execution times for the four most active job names belonging to the same user. Runtimes clearly exhibit different statistical properties, central tendencies and levels of dispersion (which can be judged by the slope of the line).



(a) Number of users in each VO (b) CPU utilisation factor

Figure 4.25: VO differentiation: by (a) user count and (b) CPU utilisation factor. VOs mostly contain just one, generic, active user which hampers lower grain monitoring of an individual's work pattern.

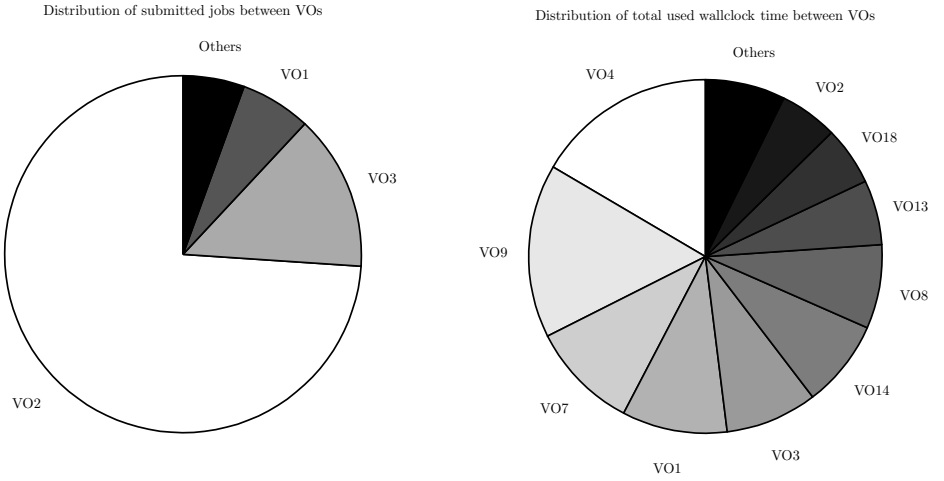
context of the job execution time predictions, and motivates the use of this job property as the basis for workload partitioning.

4.4.2 Virtual Organisation Differentiation

The value of the Grid VO meta-data is in unifying all the users from the same research project in one group. The notion of Virtual Organisations is one of the defining characteristics of the Grid, and can be of great value in workload analysis as it is likely that computing demands within a research project will be similar and distinguishable from those of projects in other fields.

However, from the bar plot of the number of users in each Grid VO, shown in Figure 4.25, it became clear that in the case of the CCC there is an almost one-to-one mapping between the users and the VOs. Although data was anonimised, after consultation with the site administrators it became clear that the VO with the highest number of member users (*VO5* containing 8 users) is actually a generic VO whose members are also included in other VOs, and that *VO6* with 3 users is in fact the system administrator VO running occasional maintenance jobs. In remaining VOs with more than one member user, a common observed practice is for only one user to submit jobs. This generic approach, whether caused by the administrative difficulty in obtaining access to the CCC facility, or by some other external factors has a detrimental effect on the ability to analyse the workload in more detail, but should not be common practice in commercial utility Grids.

The same Figure re-examines the CPU utilisation statistics by grouping the jobs according to the owning VO. The same high level of overall application efficiency is confirmed, with the small difference to the average value reported in Figure 4.23(b) due to the mentioned membership of some users in multiple VOs.



(a) Job count (b) Total wallclock runtime

Figure 4.26: VO differentiation: by (a) submitted job count and (b) total wallclock execution time. Due to the one-on-one mapping between VOs and users, the plots reveal no additional information over user differentiation ones.

The imbalance between the number of jobs submitted by a VO and the actual execution time of these jobs, shown in Figure 4.26, is very similar to the user plot given earlier. Since it was established that the User and VO job properties convey the same information, it became redundant to separate the workload with respect to both of them. All subsequent analyses will only include the reference to the Grid VO.

The differentiation of the job execution time profiles between the different VOs is evident from the comparison of the distribution functions of the top four VOs by job count given in Figure 4.27. The runtimes exhibit different statistical

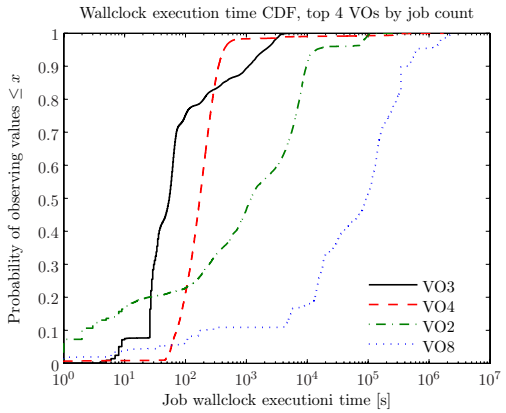
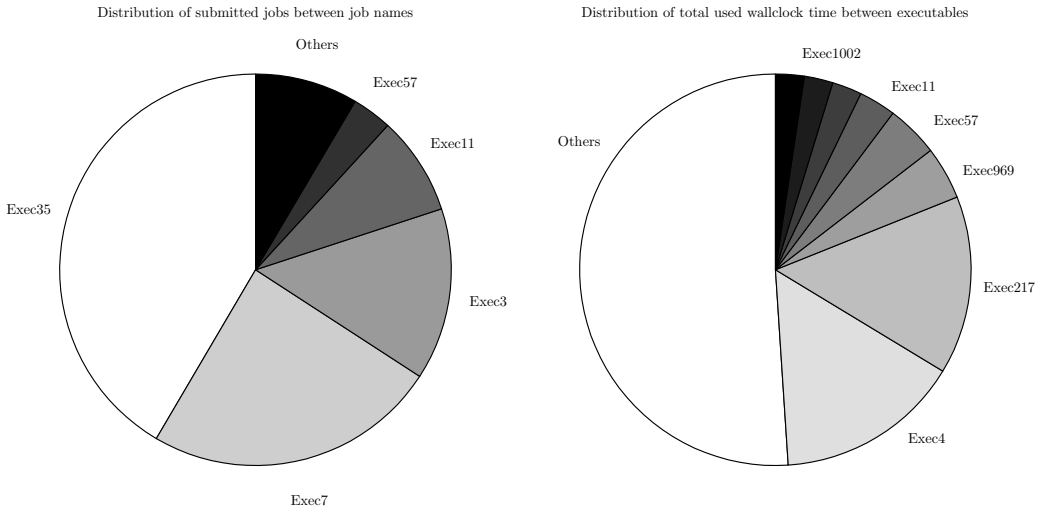


Figure 4.27: VO differentiation: comparison of distribution functions of job wallclock execution times for four most active VOs. Runtimes clearly exhibit different statistical properties, central tendencies and level of dispersion (which can be judged by the slope of the line).



(a) Job Count (b) Wallclock Time

Figure 4.28: Job name differentiation: by (a) submitted job count, and (b) total wallclock execution time. High proportion of compute time used by a mix of other names is attributed to the submission of single-use job names.

properties, central tendencies and levels of dispersion depending on which VO they belong to. Much in the same way as the submitting user, this information can be exploited to partition the workload into more predictable domains.

4.4.3 Job Name Differentiation

The wider Grid community* is still debating on how to positively and globally identify a Grid job and all of its constituent tasks. This is an important issue in the Grid workflow creation and management, and would certainly lead to more granular monitoring data. As it is, the CCC simply records the name of the executable the user has submitted to the queue. While this data is anonimised in the trace, system administrators have observed user's tendency to use generic shell scripts and wrappers to prepare the environment and launch their applications. This practice reduces the value of the job name differentiation as multiple different applications may be recorded having the same job name in the accounting file. Equally problematic is a somewhat rarer practice of assigning a unique job name for each application run.

Figure 4.28 shows the proportion of the total job submissions and the total wallclock execution time attributed to each of the job names. The job count distribution is dominated by only four, frequently submitted, generic job names, but more than 50% of the total cluster time was devoted to executing a mix of different job names. Clearly, most submitted jobs are not the most computation-

*Open Grid Forum Workflow Management Research Group and Usage Research Group

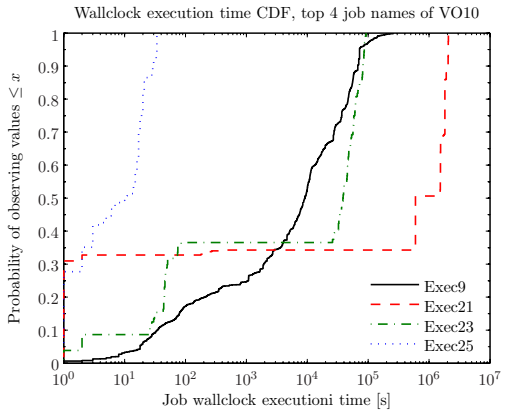


Figure 4.29: Job name differentiation within the same VO: distribution functions of the job wallclock execution times for four most submitted job names from *VO10* is shown. Runtimes clearly exhibit very different statistical properties which could not be fitted using a single, universal model.

ally expensive, and the use of one-off job names further spreads the distribution of overall runtime attributed to each job name.

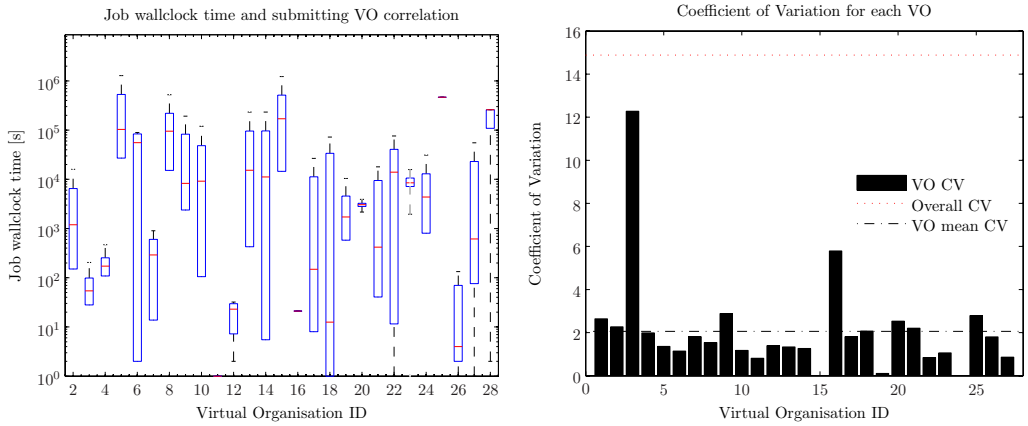
An example of the significant differentiation of the job execution time distributions between the job names submitted from the same VO is shown in Figure 4.29. Although all belonging to the same VO, different job names exhibit a very different execution pattern: two show very well defined modal runtimes while the other two are characterised by an almost log-linear runtime distribution but at very different scales. The ability to differentiate between these jobs, increases the accuracy with which their future execution time can be predicted and motivates the inclusion of the job name property as one of the workload partitioning metrics.

4.5 Correlations with Job Execution Time

The preceding section has demonstrated the diversity of the workload and the differentiation between its constituent groups of users, VOs and applications. It has also hinted at the reduction in variability achievable through partitioning the workload around several "pivot" job properties. The purpose of this section is to quantitatively and rigorously establish whether such functional dependence between the job execution times and some of it properties exists.

As the majority of the job properties are logical values, the usual correlation coefficient measures cannot be applied. The analysis will therefore be based on applying the correlation ratios, the measure of the statistical dispersion within individual categories and the dispersion across the whole population or sample, to establish the functional dependence between the job execution time and its properties.

The purpose of the following boxplots was to assist the reader in visualising



(a) Boxplot (b) Coefficient of variation

Figure 4.30: Correlation of wallclock execution times and originating VOs: (a) boxplot of runtimes for each VO, and (b) coefficient of variation for jobs belonging to each VO. Mean CV value for jobs grouped by their originating VO is many times lower than the CV value of the entire workload.

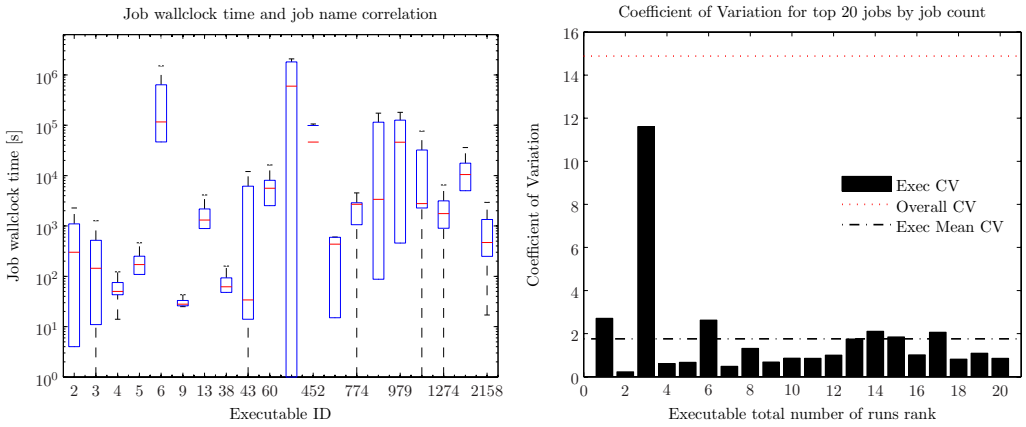
the location and dispersion of the execution times for a certain category. As such, their extreme outlier values were removed for increased legibility. The calculation of the CV values for all partitioning metrics has, of course, included all relevant jobs.

4.5.1 Job Meta-data

The correlation effects between the submitting Grid VO and the job execution time are shown in Figure 4.30. The boxplot shows the robust measures of the central tendency and dispersion for all the jobs belonging to a certain VO. The bar plot compares the coefficient of variation of the entire workload with the CV values of the jobs belonging to the individual VOs. Clearly a very significant reduction in the dispersion of the execution times has been achieved by grouping them according to the submitting VO: the mean CV by VO is 2.06 compared to 14.88 for the entire trace.

Additional benefit of this approach is the ability to recognise the high variability jobs before they begin executing (through a combination of their properties and meta-data) and take appropriate scheduling action. Such jobs could be segregated and run on dedicated best-effort nodes, or an alternative Grid economy policy may apply to them.

The boxplot in Figure 4.31 shows the medians and inter-quartile ranges of the twenty most submitted jobs in ascending ID order. Although the execution times of some jobs remain very widely dispersed, the variability of most of them is substantially decreased. The bar plot in the same figure testifies to this by showing the CV value of all the runs of the top twenty most submitted jobs ordered by their rank. For all but one job name, the CV value is around 2



(a) Boxplot (b) Coefficient of variation

Figure 4.31: Correlation of wallclock execution times and job names: (a) boxplot of runtimes, and (b) coefficient of variation of 20 most submitted jobs.

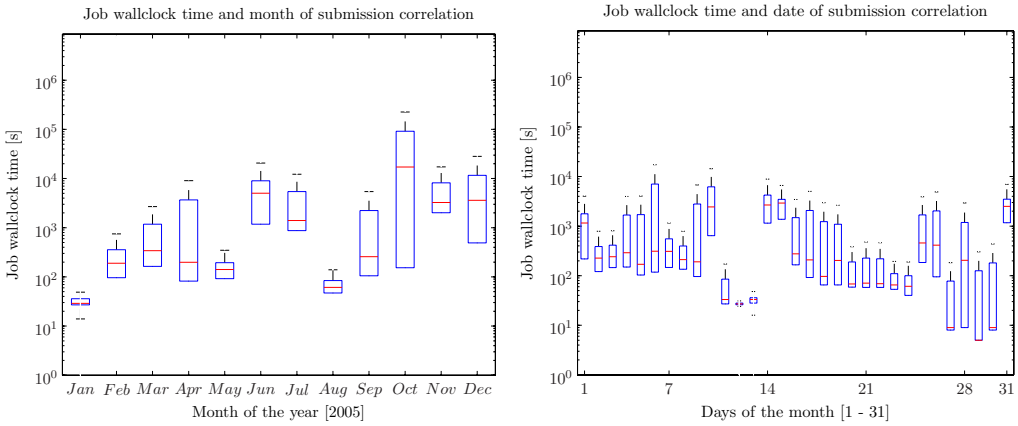
or less, with the mean CV of 1.63 compared to 14.88 of the whole workload. Clearly, there is significant correlation between the job names and their wallclock execution times.

4.5.2 Job Temporal Properties

The correlation of the time of job submission and its execution duration was already mentioned in the analysis of the runtime cyclic patterns. In Figure 4.17 on page 85, the mean execution time values were used, and have shown significant levels of variation. As the arithmetic averages can be influenced by the outlier values, the correlation analysis of the execution and submission times was repeated on several scales (year, month, week and day) using the robust inter-quartile ranges and box plots.

Figure 4.32 shows the location and dispersion of the job execution times according to the month, and the calendar day of the month, in which they were submitted. The discussion of the cyclic behaviour given earlier has concluded that the value of these two seasonal properties is limited (the yearly cycle is too long and the calendar day of the month dominated by the weekly pattern), but the reduction of variability is still evident, especially between the months of the year.

The effect of grouping the jobs according to the weekday on which they were submitted on the reduction of the average CV value is shown in Figure 4.33. The boxplot shown in (a) reaffirms the previous findings that Fridays see the longest running jobs being submitted, while the mid-week jobs are the shortest. It also points to a high variability of the jobs submitted on weekends, with the inter-quartile range for Saturday running as low 10 seconds. The modest reduction of the average CV value, 14.14 compared to the overall 14.88, shown in the bar plot

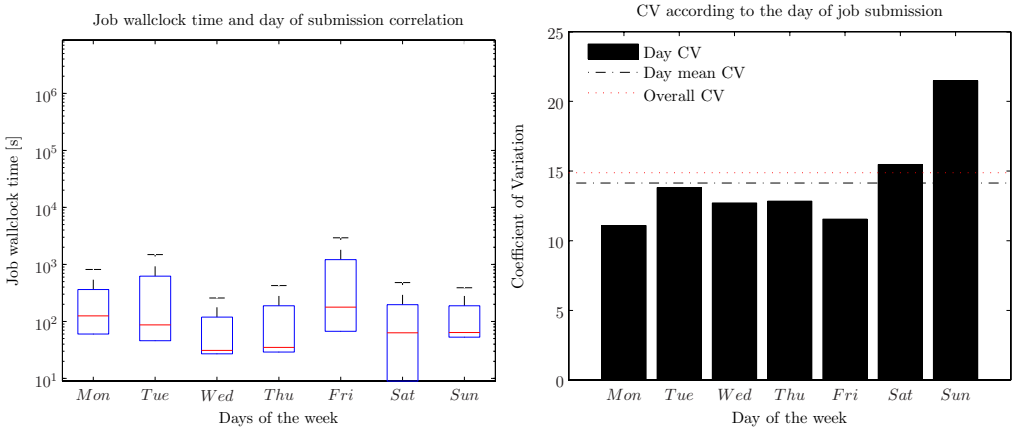


(a) Yearly boxplot (b) Monthly boxplot

Figure 4.32: Correlations of the job wallclock execution times and the job submission time on: (a) yearly and (b) monthly scales. Despite the fact that the dispersion of the job runtimes is reduced, the yearly and monthly cycles are not suitable for predicting future job execution times.

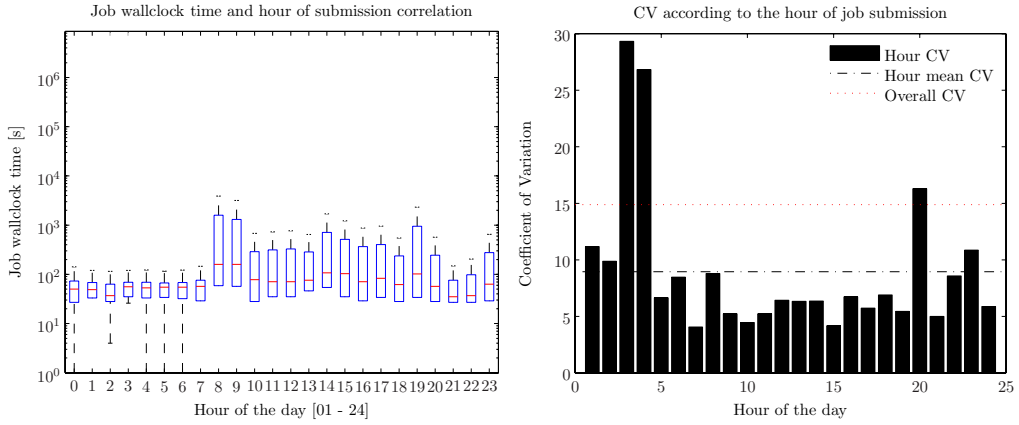
in (b), is mostly due to the high coefficient of variation of the weekend jobs. The natural reason for such high variability it the user's tendency to use the weekend to "experiment" by submitting new jobs or simply running a mixed workload that has perhaps failed during the week or needs to be re-done. Another factor, as previously discussed, is the instability of the CV measure for the series with small means.

The correlation between the hour of the job submission and its execution time is evident from the reduction of the runtime variability shown in Figure 4.34. The boxplot in (a) shows a clear difference between the execution times of jobs sub-



(a) Weekly boxplot (b) Weekly CV reduction

Figure 4.33: Correlations of the job wallclock execution times and the weekday of the job submission. A modest reduction of the CV value was mostly influenced by the very large variability of the jobs submitted over the weekend.



(a) Hourly boxplot (b) Hourly CV reduction

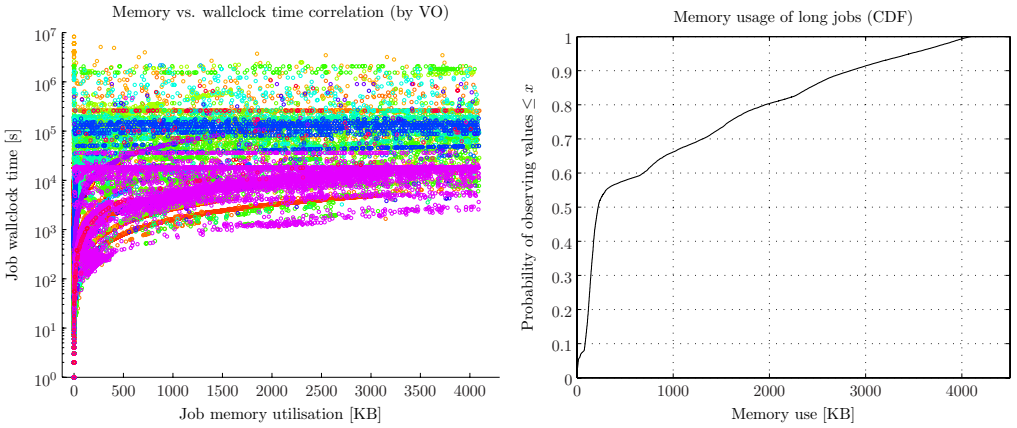
Figure 4.34: Correlations of the job wallclock execution times and the hour of the job submission. Daytime CV values are around 5, while the much higher off-peak variability raises the average CV value to 8.95.

mitted in daytime and of those submitted during the night. The early morning, lunch hour, and late afternoon peaks in execution times are again prominent. The bar plot in (b) reveals that by grouping the jobs according to the hour of their submission reduces the average CV value of the daytime jobs to around 5 and to 8.95 for the entire 24 hour period. While higher variability in the late afternoons is expected, as numerous users submit their jobs for the anticipated overnight execution, the very high CV values observed in the early morning hours are caused by the small mean execution time of the jobs submitted between 2am and 6am. As it can be seen on the boxplot, the whisker for those hours extend down to 1 second in duration indicating a low mean and the instability in the CV measurement leading to high values.

4.5.3 Memory Usage

A common assumption that the longer running jobs would require more memory is only partially supported by the analysis of the CCC trace. Figure 4.35(a) shows a run sequence plot of the execution time of all trace jobs versus their total allocated memory, color coded according to the Grid VO owning the job. Probably the only undisputed fact, supported by the lack of data points in the lower right part of the plot, is that short running jobs do not allocate large amounts of memory. This, however, only holds true for the jobs running up to about an hour as about 95% of all the jobs in the trace does. Jobs running for longer than that are allocating memory from almost 0 to the maximum 4096 MByte value. Additionally, the plot indicates that a low memory utilisation does not necessarily imply short execution time: a significant number of data points are present in the upper left part of the plot.

Observing this effect, a question arose whether the longer running jobs have



(a) Runtime - Memory use correlation (b) Memory use for job runtime ≥ 1hr

Figure 4.35: Correlation of wallclock execution times and total allocated memory, shown in (a), indicates that jobs running for less than one hour do not allocate large amounts of memory. The CDF plot of memory allocation of longer running jobs, shown in (b), shows two differently sloped but very linear modes of memory usage.

a certain preference for allocating specific amounts of memory. Figure 4.35(b) plots the memory usage distribution function for jobs executing for more than one hour. Around half of these jobs allocate less than about 300 MBytes of memory, while the other half allocates between 300 MBytes and the maximum installed amount. Interestingly, both segments of the CDF plot are very linear indicating lack of modality or preferences for any specific value.

The correlation between the memory utilisation and the job execution time was established using the Spearman's rank correlation coefficient returning the value $\rho = 0.75$. Such a result indicates a significant positive correlation between the amount of allocated memory and the wallclock execution time, a property which has previously been studied in the literature [186, 25, 23, 187] but on which no consensus was made as it seemingly differs between the workloads.

Although job memory requirements can be an important criteria in the resource selection part of the scheduling process, its value in the context of ex-ante* prediction of job execution times is limited. The amount of memory that the job will allocate at its start is not known while the job is queueing, and hence cannot be used to increase the accuracy of the execution time predictions. Even if the job can be re-scheduled during runtime, processes rarely allocate all of its required memory at once, so that the total amount of memory a job has used is not known before it finishes its execution.

*Latin for "before the event". In models where there is uncertainty that is resolved during the course of events, the ex-ante predictions are those that are calculated in advance of the resolution of uncertainty.

4.6 Locality of Sampling

The workload characterisation so far presented made passing remarks about the overall evolution of the relevant metrics through time, and the associated changes in their statistics. Workload properties for those Grid VOs that have run more often and over a longer time period seem to have higher dispersion and more variance than those whose jobs are run over a short time scale.

The purpose of this section is to establish the level of sustained changes and transient spikes in the workload properties so that an appropriate adaptive technique could be used to handle these features and increase the accuracy of the predictive scheduling models. The notion of the sampling locality, introduced in the methodology section of this chapter (see page 73) will be tested on the four metrics of primary interest to the job scheduling: the job arrival rate, job inter-arrival time, queue wait time and the wallclock execution time.

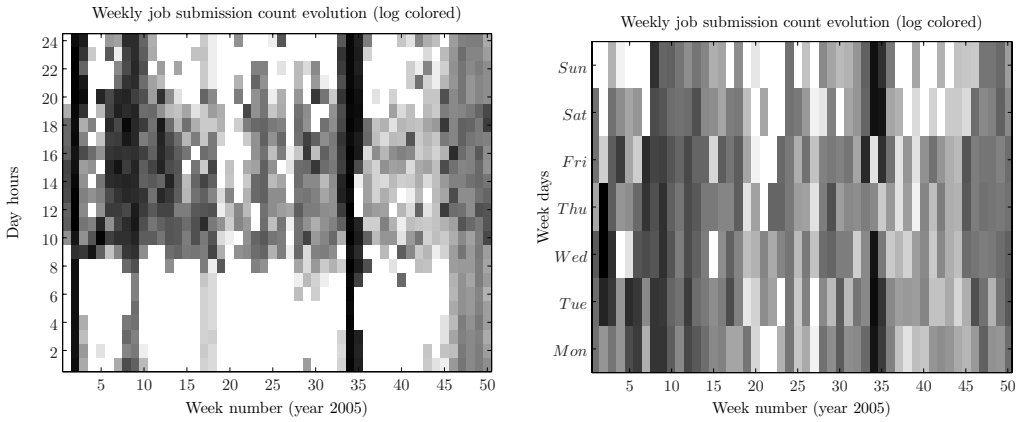
The summary analysis of these metrics was already given in the previous sections; the focus here will be on using novel statistical graphics methods to visualise the changes that the workload experiences over an extended period of time.

4.6.1 Job Count

The evolution of the daily job submission pattern, according to the hour (a) and the weekday (b) of the job submission, for each week of the year long trace is given in Figure 4.36. Previous conclusions that the majority of the jobs throughout the year are submitted during extended office hours of 8am - 8pm is clearly confirmed. The features of the slower job submission tail-off in the evening, the lack of post-midnight jobs, and the abrupt morning rise in the job submissions are also clear. Despite this overall pattern, reading the plot along the x axis at a constant y value (being the count of the jobs submitted at 10am, for example, in each weeks), significant variations can be observed.

The plot also shows that the usual pattern has been severely disrupted on a number of occasions. During the last four weeks of the trace the job arrival rate is almost constant throughout the day, and in weeks 2, 9 and 34 a very large number of jobs was submitted during the entire 24 hour period.

These features correlate with the plot of the weekday job submission counts shown in the adjoining plot 4.36(b). Week 2 is characterised by a high level of job submissions on Wednesday and Thursday, while in the week 34 a large number of jobs was submitted on all days except Thursday and Friday. This plot also explains a somewhat counter-intuitive result of the weekend job submission rate being on the same level as the weekday one, previously reported by Figure 4.7 on page 78. Week on week, Saturdays, and especially Sundays, see a low number of submitted jobs, but the overall count is raised by several weekends when a large number of jobs have been submitted.



(a) Daily cycle (b) Weekly cycle

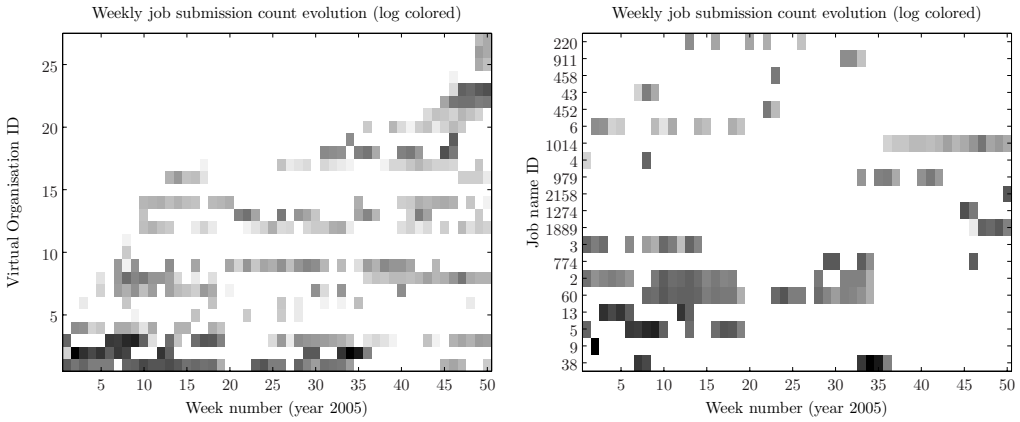
Figure 4.36: Locality of sampling: number of jobs submitted, shown by gray levels on a logarithmically scaled range from 0 to 10^4, as a function of their submission time on (a) daily and (b) weekly basis in each week of the year. Large variations suggest a single, static model fitted to the entire trace is not likely to give acceptable results.

A similar analysis could be done with respect to the evolution of the workload according to the submitting VO and the job name. Figure 4.37(a) shows the number of submitted jobs in each week by the members of each Grid VO. The fact that the plot has its data on one side of the imaginary $y = x$ line indicates that the VOs have been created as new users joined the CCC community, and that the number of the VOs has grown throughout the year.

The sporadic activity of the users is clearly visible on this plot: periods of high activity are followed by a complete lack of job submissions, after which many users return to the system and submit some more jobs. These features are consistent with the expected user workflow which is made up of preparatory periods in which the jobs are test run, followed by the "production" runs which can take several weeks of heavy job submissions. The subsequent lack of activity could indicate the user is analysing the results of submitted jobs and preparing for further job submissions.

Reading the plot vertically (observing all Grid VOs in one week of the year) shows that only a fraction of all CCC users is active at any given time, and that the VOs making up the workload in any give week is changing. The VOs are also likely to be in the different stages of their workload cycle with some being in a test phase, and some in a production phase characterised by the heavy job submissions.

The number of weekly submissions for the twenty most often run jobs is given in Figure 4.37(b). Some jobs are submitted only within one or two weeks, while some are executed in many disjoint sessions lasting between one and ten weeks. The submission rate over those periods tends to be fairly constant as well. This insight into the submissions cycle for each application could be used in the job



(a) VO (b) Job name

Figure 4.37: Locality of sampling: number of jobs submitted, shown by gray levels on a logarithmically scaled range from 0 to 10^5, originating from a specific VO (a), or having a specific job name (b)(only top 20 jobs by submission count shown). Both plots reveal epochal behaviour with periods of high and low activity.

admission control and potentially in some form of advanced reservation system.

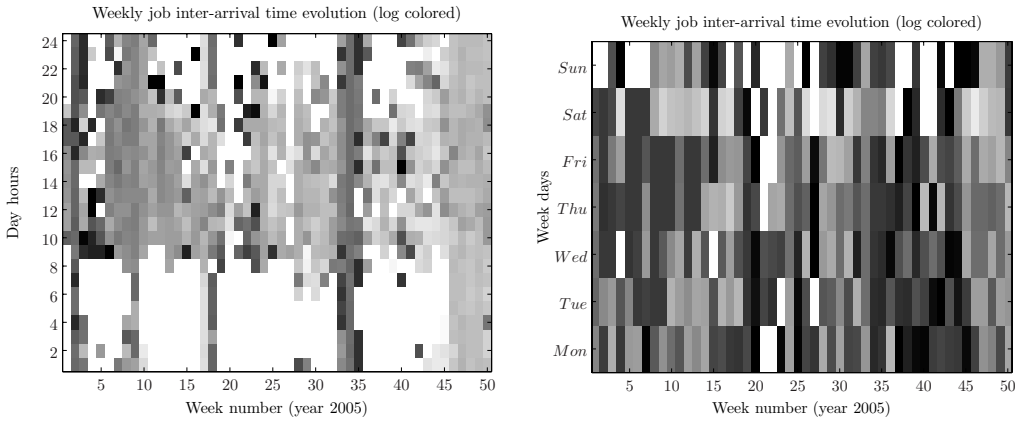
4.6.2 Inter-arrival Time

The weekly fluctuations of the job inter-arrival times, as a function of the hour and the weekday of the job submission are given in Figure 4.38. The colouring scheme for the inter-arrival time plots has been inverted, with the lower mean values taking darker shades and thus indicating a higher rate of job arrivals ("hotspots").

The hourly plot, shown in (a), is characterised by a period of almost no activity between midnight and 8am, as well as the already mentioned periods within which job submissions were present around the clock (weeks 33-34 for example). It is now clear that a very different job arrival pattern has taken place in the last four weeks of the trace, as the inter-arrival times are almost uniformly spread out throughout the day for an extended period of time. This could indicate an automated submission of jobs according to some policy, or an administrative arrangement that was supposed to run over the perceived off-peak period of college closures (the Christmas break period).

The plot also shows a number of instances of very short inter-arrival times which mostly occur at the beginning of the workday or at some point in the late afternoon or evening. When considered together with the already established tendency to submit more and longer running jobs at this time, it seems that the users are sending prepared job batches for execution in the morning and before leaving offices in the evening. The anticipation of such behaviour could be very valuable to the predictive deadline scheduler.

The weekly fluctuations of the inter-arrival times according to the weekday of



(a) Hourly (b) Daily

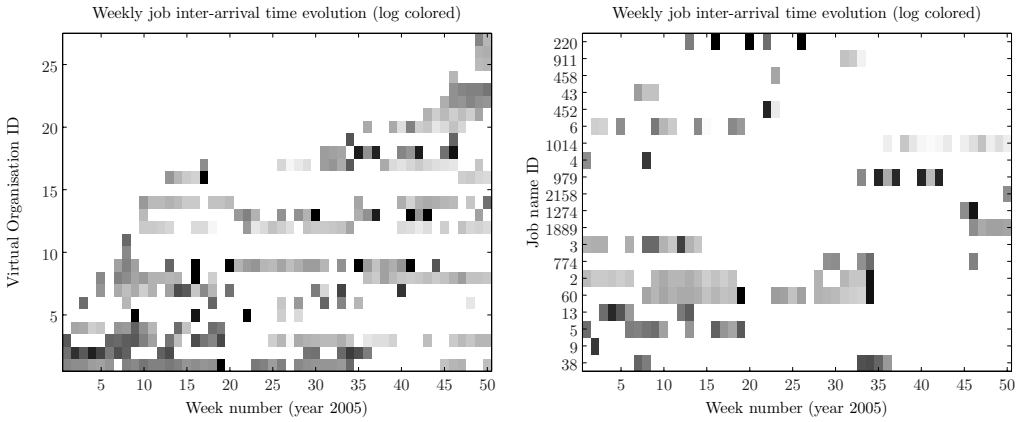
Figure 4.38: Locality of sampling: job inter-arrival times, shown by gray levels on a logarithmically scaled range from 0 to 10^4, as a function of their submission time on (a) hourly and (b) daily basis in each week of the year. Apart from seasonality patterns previously noted, the plots show a longer term changes in the job inter-arrival times, as well as isolated "hotspots" of bulk job submissions.

submission are plotted in Figure 4.38(b). When present, the job submission on the weekends is characterised by very small inter-arrival times and could indicate the user's intention to submit a set of already prepared jobs for the execution before the perceived Monday rush. The plot also shows a high degree of variance, both within each week, and between the same days in different weeks.

The pattern of the job inter-arrival times partitioned according to the Grid VO and the job name properties, Figure 4.39, shows much the same features as previously observed in the job count plot. The sporadic submission of the jobs by the facility's users is evident, and the reuse of the job names is also present. Interestingly, the bulk submission of the jobs, leading to very short inter-arrival times and dark patches on the plot, are either preceded by the periods of moderate activity, or are followed by an extended periods of no job submissions. This insight, strengthened by the conversations with some of the users, again points to the epochal nature of the workload in which the jobs are prepared and tuned before a large batch is submitted for execution.

4.6.3 Queue Time

The weekly variations in the job mean queue wait times, as a function of their hour and weekday of submission, are shown in Figure 4.40. The attention is immediately drawn to the week 34 in which all of the submitted jobs exhibit a very long queue delay. Cross-referencing the two plots, it is clear that the delay was caused to all of the jobs submitted throughout the 72 hour period between Wednesday and Friday of the week 34. One of the likely reasons for such a long delay would be the blocking of the queue by several very long running



(a) VO (b) Job name

Figure 4.39: Locality of sampling: job inter-arrival time, shown by gray levels on a logarithmically scaled range from 0 to 10^4, originating from a specific VO (a), or having a specific job name (b)(only top 20 jobs by submission count shown). Both plots reveal epochal behaviour with periods of high and low activity, as well as specific "hotspots" where a large number of jobs has been submitted in very short period of time.

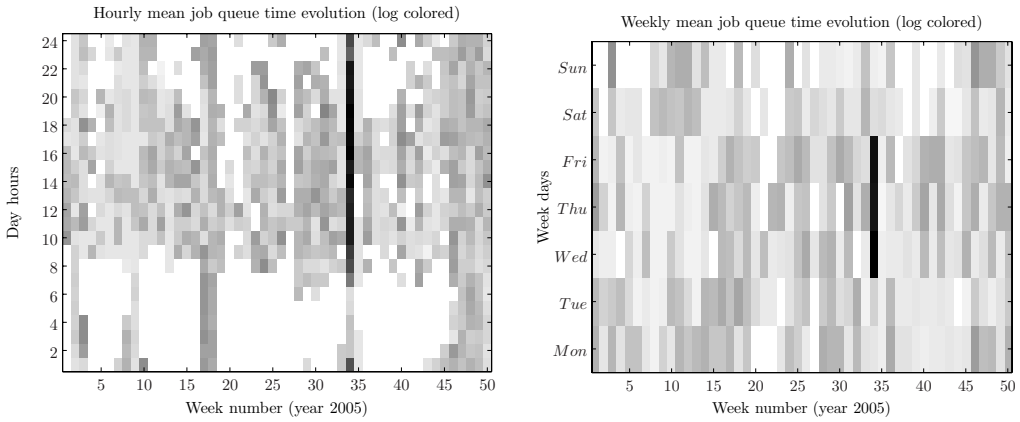
jobs. Although the predictive scheduling techniques could not completely solve these kind of problems, the slack factor* of the user requested deadline serves as a dynamic prioritisation measure and could help the owners of the shorter, but more urgent jobs to jump the queue.

Apart from this unusually long queueing time, Figure 4.40 re-confirms that the majority of submitted jobs experience generally low queue wait times. Weekend, late night and early morning jobs are least delayed due to queueing, while the queueing time of the remaining workload is mostly influenced by the overall utilisation of the facility and the fullness of the scheduling queue.

The plot in Figure 4.41 shows the weekly variation of the queueing times based on the job's owning Grid VO and the job name (only 20 most submitted job names are shown). From (a), it seems as all VOs experience the entire range of the queuing times, thus indicating the fairness of the scheduler and the lack of any special administrative policies prioritising jobs submitted by a certain VO. The level of the queue delay seems to be, at least to some extent, influenced by the number of active users in any given week. Low activity weeks, such as week number 20 when only two VOs are active, generally see shorter queue wait times.

The same conclusions can be draw from the plot of the queuing delay experienced by the top 20 most submitted job names given in Figure 4.41. No prioritisation seems to be taking place with the jobs experiencing longer queueing times when more concurrent applications are running.

*The ratio of the actual execution time and the time between the job submission and the requested deadline



(a) Hourly (b) Daily

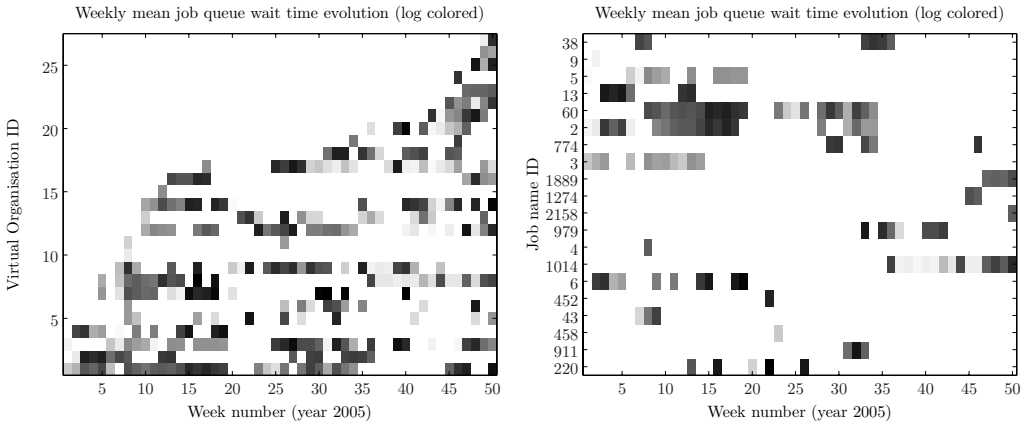
Figure 4.40: Locality of sampling: job queue wait time, shown by gray levels on a logarithmically scaled range from 0 to 10^6, as a function of their submission time on (a) hourly and (b) daily basis in each week of the year. The daily and weekly cycles are again evident as jobs submitted off-peak tend to queue less. Jobs submitted mid-week 34 have for some reason experienced very long queueing times.

4.6.4 Wallclock Execution Time

The weekly evolution of the job wallclock execution time, plotted as a function of its hour and weekday of submission, is given in Figure 4.42. A significant level of variance throughout the trace is present, with certain weeks seeing the submission of some very long running jobs. While it may be difficult to distinguish the overall features and tendencies, as given in Figure 4.17, the high and the low intensity phases of the workload are clearly visible. A strong job campaign took a break around week 20, followed by another 4 weeks of significant workload, and then a period of generally shorter running jobs. These features are also evident on the weekday plot. The last five weeks of the workload stand out again with continuous job submission throughout the day, but even here, the tendency to submit longer running jobs between 10am and midnight is present.

Throughout the day, the busiest hours are 9am to 8pm with specific execution time "hotspots" in the early morning and the late afternoon. Looking at the weekdays plot, the lower length of the weekend job executions is evident. Saturdays and Sundays generally see the submission of very short running jobs. On several occasions, such as between weeks 5 and 10, a specific job campaign execution solely on weekends seem to have taken place. Such behaviour could be the effort of the users to do some load balancing themselves and try to obtain better performance from the facility by submitting at the obvious off-peak hours. Hardly a better motivation can be had for an economy and deadline based approach to system balancing and yield management.

The reduction in the variability of the job execution times achievable through



(a) VO (b) Job name

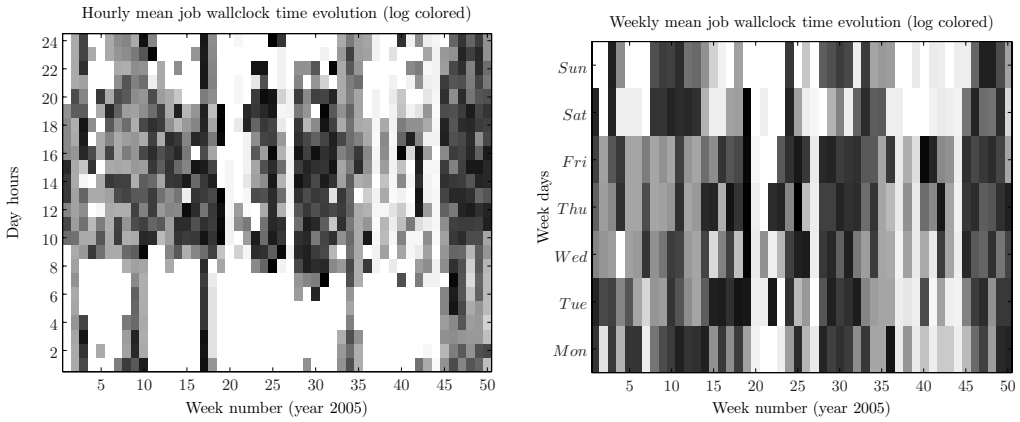
Figure 4.41: Locality of sampling: job queueing time, shown by gray levels on a logarithmically scaled range from 0 to 10^5, originating from a specific VO (a), or having a specific job name (b)(only top 20 jobs by submission count shown). Similar job queueing times for all VOs hint at the lack of specific VO-level prioritisation, but the number of active VOs at any given time has an influence on the queue waiting times.

workload partitioning based on the job's properties is again evident from the plots in Figure 4.43. The mean execution times, and hence the intensity of each plot patch, differ substantially between the VOs, in (a), but are quite consistent within one VO. The value of sampling locality is demonstrated on the example of the *VO3*. Jobs run by this VO clearly have two modes of the execution length before and after week 25. Averaging over the whole trace period would yield a model not representative of either of these periods, while they clearly show little dispersion and could be predicted quite well. These modes are indicative of the evolving nature of the workload which has perhaps moved onto using a different data set, different application or altogether a different research objective.

Similar characteristics are evident in the job name plot, Figure 4.43(b). The variation of the execution times between different job names is much greater than between the different runs of the same job. The modal characteristic of the execution time present when the workload is separated by using the VO job property is here not evident. A likely reason is that a significant change in the job's application, workflow or analysed data would be most likely followed by the change in the job's name by the user.

4.7 Chapter Summary

The chapter has presented and exhaustive characterisation of a year long trace sourced from a production Grid installation. The analysis has concluded that in a multi-purpose, utility style scenario, the Grid is likely to service numerous users with varying resource requirements, workflow characteristics and performance



(a) Hourly (b) Daily

Figure 4.42: Locality of sampling: job wallclock execution time, shown by gray levels on a logarithmically scaled range from 0 to 10^6, as a function of their submission time on (a) hourly and (b) daily basis in each week of the year. Plots indicate significant and sustained changes in the length of job execution as well as periods of high and low activity. Large variations suggest a single, static model fitted to the entire trace is not likely to give acceptable results.

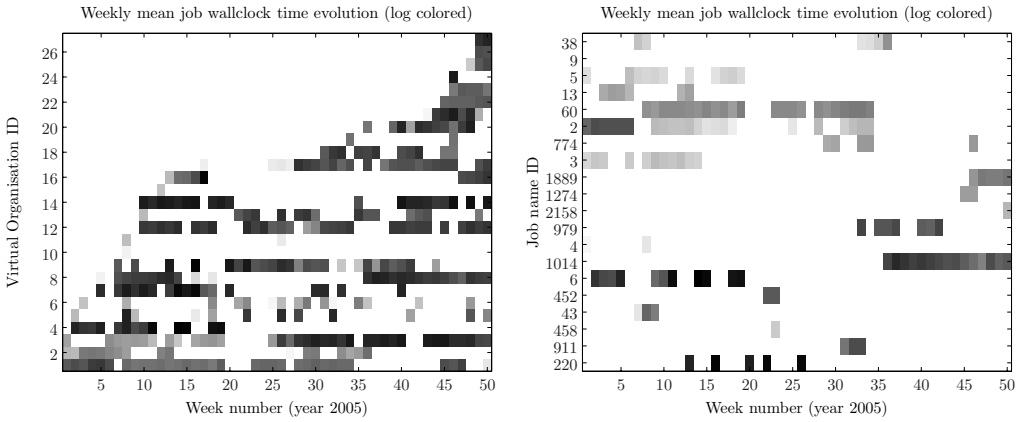
expectations. This diversity leaves an opportunity for the probabilistic resource management to maximise the usage of the installation while delivering required service levels to the users.

The workload analysis has focused on the job arrival process, queueing time, job wallclock execution time and the memory utilisation. Overall, all but the memory utilisation were found to follow a weekly and daily cycles, have a very high coefficient of variation and exhibit strong self-similarity and long-tail properties. The values of the job inter-arrival times and the execution times were also distributed in a log-normal fashion. The summary of these findings is given in Table 4.4.

The characterisation paid special attention to the diversity of the workload and the differences between the primary metrics for the jobs belonging to different users and VOs, or having different job names. The findings pointed to some important aspects of the workload and can be summarised as follows:

- Due to the administrative policies, the mapping between the VOs and their member users was almost one to one. Where a VO had more than one user, only one would submit jobs. Such practice rendered the submitting user job property useless as it contained no more information than supplied by the job's owning VO field.

- A familiar 90-10 split was observed on the number of submitted jobs: jobs of the three most active VOs accounted for almost 95% of submissions. Same was not true for the distribution of the total wallclock time of the



(a) VO (b) Job name

Figure 4.43: Locality of sampling: job wallclock execution time, shown by gray levels on a logarithmically scaled range from 0 to 10^6, originating from a specific VO (a), or having a specific job name (b)(only top 20 jobs by submission count shown). The plot shows jobs from the same VO to run for similar amounts of time, while differing significantly from those submitted by other VOs.

facility which was split much more evenly. Clearly the more frequently run jobs execute for much less time than the sporadically submitted ones.

- Execution times of the different job names submitted by the same VO vary significantly between each other, but are very autocorrelated and similar to their previous runs. The distribution of the runtimes for the top 20 most submitted jobs, grouped by their job name, has been found to be very narrow and deterministic.

- The number of different job names submitted by each VO seems to be modal: the majority of VOs submit all the jobs with the same, generic name; several VOs use up to a hundred different names while 4 VOs apparently use a unique name for almost each submitted job. Considering the importance of uniquely identifying the submitted job or application, more

	Cyclic period					
	Weekly	Daily	Log-normal	CV	Long-tailed	Hurst
Arrivals	●	●	●	36.81	● $(> 3s)$	0.85
Queue time	●	●		2.85	● $(> 10^3 s)$	$1 \approx 1$
Runtime	●	●	●	14.88	● $(> 10^2 s)$	0.87
Memory				3.08	◐	

Table 4.4: The summary of the general properties of the four primary metrics analysed in the workload characterisation study. The weekly and daily cycles, large CV values and the strong self-similarity were common. The inter-arrival times and execution time were also distributed in a log-normal fashion.

granularity would significantly increase the ability to statistically predict the job execution times.

The correlation between the job execution time and the temporal and meta job properties have been studied by comparing the variability of the dataset grouped according to a specific "pivot" property to that of the entire trace. A reduction in the coefficient of variation is indicative of a functional dependence between the job runtime and studied property. The summary of the achieved results is given in Table 4.5

Grouping the jobs according to their job name and the submitting VO has given very good results. The use of the temporal job properties, such as the hour or the weekday in which a job was submitted, has also produced a reduction in the variability of associated execution times. A more limited benefit of using these two temporal characteristics was caused by two main reasons.

Firstly, the CV measurement becomes highly sensitive to the changes of the standard deviation as the series mean approaches zero, as was the case for some off-peak, mid-night and weekend periods in which very few short running jobs were submitted. The overall effect of these high values was further increased by the use of the arithmetic mean as the measure of the central tendency. Secondly, the temporal characteristics were envisaged as a supplemental, highly granular, job differentiation metric to be used in conjunction with the other job meta-data. An example of such use, and its benefits, will be presented in Chapter 5.

The correlation between the job's total memory utilisation and its execution time has been calculated using the Spearman's rank order coefficient. The indicated substantial positive correlation could not be used in predicting the length of the job's execution as the amount of the memory used is only available once the job has completed.

	Coefficient of Variation (CV)		
	Mean	% of Overall	Spearman's
Overall	14.88	100.00	
VO	2.06	13.84	
Job name	1.63	10.95	
Daily	14.14	95.03	
Hourly	8.95	60.15	
Memory	3.08		0.75

Table 4.5: The summary of the correlation of the job execution time and its meta and temporal properties. Higher reduction in the CV value indicates stronger functional dependence. The correlation of the job memory utilisation and its execution time was calculated using Spearman's rank order coefficient.

The workload characterisation study has also dealt with the presently poorly

researched topic of the locality of sampling and long term evolution of the workload properties. The purpose was to distinguish which properties of the workload are constant and which tend to change over time, and thus assist in properly engineering the adaptability of the job execution time prediction model.

The four most important workload metrics, the submitted job count, inter-arrival times, queueing time and the wallclock execution time, were analysed using a novel plotting technique emphasising the differences between the job as a function of their temporal or meta properties, and the evolution of these properties on a weekly basis. The findings can be summarised as follows:

- The presence of daily and weekly cycles, usage patterns and seasonal variations was observed in all four metrics for the entire duration of the workload trace.
- Although these properties were constantly present, their long term evolution and fluctuations would cause a model based on a static training set of "older" data to demonstrate a significant lack of fit.
- The motivation is therefore strong for a dynamic and adaptable approach, one that is able to use the insight of the global perspective while at the same time adapting to the local fluctuations and track them in the prediction model.
- The graphical technique used helped in confirming that the workload was characterised by the epochal nature of the job submission with only a limited number of users and applications active at any one time. The behaviour of individual users was also "on/off" with periods of activity followed by the periods of no activity.
- The analysis has also identified occasional "hotspots" of highly increased rates of job submissions or prolonged execution times of jobs. Such events occurred often enough to be represent a feature of the workload, and as they could not simply be filtered out a robust system for their handling is necessary.
- The evolution of the job execution time has revealed changes in the statistical properties of the jobs submitted by a specific VO or with a specific job name. A statically parametrised prediction model would obviously struggle with such changes.

Overall, the characterisation study has answered the questions relevant to modelling and predicting job execution times based on historical information. Its purpose was not to specifically identify most suitable models for representing its various properties, which is the common goal of the studies supporting generative models, but to explore the relationship between the job properties available to

the scheduler prior to the running of the job and their influence on its execution time.

Chapter 5

Job Execution Time Forecasting

Prediction is very difficult, especially if it's about the future

— Nils Bohr, Physics Nobel laureate

Following the in-depth analysis of the CCC Grid workload, the predictive work presented in this chapter will use those findings as a basis for delivering ex-ante forecasts of the execution times of queued jobs based only on their historical performance and associated temporal and meta-properties. A heuristic approach to grouping similarly behaved jobs is complemented by self-parametrised, time-series forecasting models to create an autonomous prediction engine. The performance of the system was tested using a real-world Grid workload, and has clearly shown the value of the more advanced prediction algorithms, the proposed job partitioning approach and the novel use of temporal job properties.

The chapter opens with Section 5.1 by reiterating the motivation for job execution times predictions and the scope of the work. Experimental methodology and details of specific techniques and approaches are discussed in Section 5.2. Sections 5.3 give experimental results of different scenarios, while Section 5.4 summarises the findings and concludes this chapter.

5.1 Purpose and Motivation

Predicting the job execution times is the core enabling technology for Grid deadline scheduling, and presents a distinct research contribution of this thesis. The purpose of the job runtime prediction work was to leverage the findings of the workload characterisation study and develop an engine suitable for the prediction of the job execution times. The fact that these were found to be highly

autocorrelated, and functionally dependent on a specific set of job properties, strongly supported the author's focus on the statistical time-series analysis as a forecasting model of choice.

The extent to which such a forecasting model could be made robust to the abrupt changes in the operating environment and to the outlier values in the time-series, was the subject of extensive work due to the target usage scenario of an on-line utility Grid scheduler. To asses the level of performance achievable in the production environment, a series of experiments using the actual Grid workload has been undertaken, all sharing the following two aims:

- Compare different time-series forecasting methods amongst each other and to other common prediction models and analyse their performance.
- Establish the added value, in terms of the increased prediction accuracy, of the job partitioning according to one or more job meta and temporal properties.

5.2 Specific Methodology

The primary challenges in the development, implementation and testing of the job execution time forecasting approaches were in choosing which specific modelling techniques to use, designing an autonomous parametrisation technique for those models and selecting the most appropriate error measure to compare the results. This section will present the chosen prediction methods in detail, offer an extensive justification of the selected accuracy measures and document the software and hardware set-up used for the experiments.

The section will also introduce the heuristic used for partitioning the entire workload around different "pivot" job properties leading to a reduction in the job execution time variability and an increase in the prediction accuracy. This job clustering method was developed based on the findings of the workload characterisation study presented in the thesis, but should be equally applicable to other Grid workloads as well.

5.2.1 Job Partitioning

One of the reasons for the extensive Grid cluster workload characterisation presented in Chapter 4 was to identify any seasonal variations, specific patterns and correlation of the execution time with other job properties and meta-data. The analysis concluded that a very variable job execution time series can be partitioned according to its temporal and meta-properties into subsets with substantially lower dispersion. This reduction of the coefficient of variation (CV) is a significant factor in enabling effective runtime predictions using automated statistical methods.

As the goal was to create a substantially self-managing system, identification of correlations between the job execution time and other job properties for an arbitrary workload was implemented using an automated and non-parametric approach. The full pseudo-code is given in Listing 5.1, and is further described in the following paragraphs.

Listing 5.1

```
initialise( prop_set );

for r = 1 to sizeof( prop_set )
{
  prop_perm[] += permut( r, prop_set );
}

foreach (prop_perm)
{
  if sizeof( prop_perm ) > m  //sufficient number of data points
    corr[] += compareRuntimeCV( prop_perm );
}

sort_descending( corr );
```

```
initialise( prop_set );
```

Initially, the workload history is loaded and parsed for job execution times and n job properties. The number and selection of these job properties will depend on the information collected by the specific Grid site and on the insight into workflow practices provided by the site administrator. The list of properties is stored in the `prop_set` array.

```
for r = 1 to sizeof( prop_set )
  {  prop_perm[] += permut( r, prop_set );  }
```

One or more job properties can be simultaneously used to partition the workload. For example, all jobs belonging to a certain VO could be grouped and modelled as one partition, or could further be divided into sub-groups based on the submitted job name. The above loop increments the number of job properties that will be used for partitioning from one to the maximum number of available properties n. Function `permut()` returns all permutations of `r` elements from the property set and appends them to the `prop_perm[]` array.

```
foreach (prop_perm)
{
  if sizeof( prop_perm ) > m //sufficient number of data points
    corr[] += compareRuntimeCV( prop_perm );
}
```

Each entry in the `prop_perm[]` array represents a possible workload partitioning criteria and is examined in turn. Depending on the workload, the number of job properties and their granularity, some highly selective partitions may not have sufficient number (`m`) of data points and are not further considered. Function `compareRuntimeCV()` compares the mean CV of job runtimes within a certain partition with the CV of a less specific, parent partition. For example, the mean CV of the submitting VO - Job name partitions is compared to those created by using only the submitting VO job property. A reduction in the coefficient of variation indicates a correlation between the job execution time and the property in question.

```
sort_descending( corr );
```

The correlation results, stored in the `corr` array, are sorted in descending order giving a ranked list of partitioning metrics with strongest correlation to the job execution time.

The scalability of the approach, which is essentially an exhaustive search of the job property space, is dependent on the total number of job property permutations, given by the following equation:

$$N_{max} = \sum_{r=1}^{n} P_r^n = \sum_{r=1}^{n} \frac{n!}{(n-r)!} \tag{5.1}$$

where N is the number of job property permutations, n the total number of job properties and r the number of selected job properties.

In the case of the CCC workload, the number of relevant job properties was 6 leading to the maximum number of permutations $N_{max} = 1856$, of which almost 90% did not contain any data points. Other surveyed workloads had an equally small number of recorded job properties (fewer than 10). Considering that coefficient of variation is computationally inexpensive to calculate even for large time series, and that the algorithm is run ad-hoc and off-line, the performance of the proposed approach should not be an issue.

The application of this algorithm on the CCC workload has found that job partitioning according to the Grid VO owning the job, job unique name and the calendar week in which the job was submitted is most likely to significantly reduce the level of runtime variability within each partition. These three metrics will therefore be used as the key "pivot" properties for job partitioning in all of the following job sets.

To analyse the performance of different prediction methods, and the influence that job clustering based on different job properties has on the forecast accuracy, a number of data sets was used. Consisting entirely of the actual and unchanged Grid jobs present in the CCC trace, these pre-defined job sets were needed to ensure enough training data points are available and that prediction methods can be repeatedly compared against the same benchmark. A workload set, in this

context is simply a fixed collection of real jobs partitioned using one or more of the job properties.

The following sections will present the effect of job partitioning on the variability of job execution times using direct comparison of the coefficients of variation. All plots report mean CV reduction based on the analysis of the entire workload trace.

Single Metric Job Partitions

From the analysis of correlation between job properties and its execution time, given in the workload characterisation Section 4.5, significant reduction in the dispersion of the runtime values is evident even when jobs are partitioned even using only one of the job properties. The effect this would have on the accuracy of execution time predictions is tested using the following three partition sets:

Submitting VO set contains jobs separated by the identity of their submitter. Virtual organisations which submitted less than 100 jobs in the whole year were excluded as they may not have sufficient training data for the forecasting algorithms. The representativeness of the set was not compromised by this, as those 8 excluded VOs submitted only 363 jobs altogether accounting for 0.05% of all job submissions and 1.13% of overall execution time. A plot showing the reduction in CV compared to the overall value was given in Figure 4.30 on page 96.

Job name set is a subset of the full job name set holding execution time values of the top 30 most submitted jobs. Although there was more than 2200 unique job names in the observed period, this relatively small subset (1.32% of all of the unique job names) captures 97.89% of all job executions and 60.24% of the overall execution time. A plot showing a similar partitioning for the top 20 most submitted jobs was given in Figure 4.31 on page 97.

Week of Execution set reflects the temporal locality of the data and the dependency of its dispersion on the sampling window size. In this set, the execution times are partitioned according to the week of the year in which they were submitted with no respect to their owner or any other job property. This set contains every job submitted during the observation period.

Multiple Metric Job Partitions

While partitioning the entire workload according to one of the identified job properties reduces the coefficient of variation for all examined sets, the level of reduction varies between individual groups or job names. Compound sets examine the superposition of multiple partitioning parameters that have previously been shown to reduce the variability of the data.

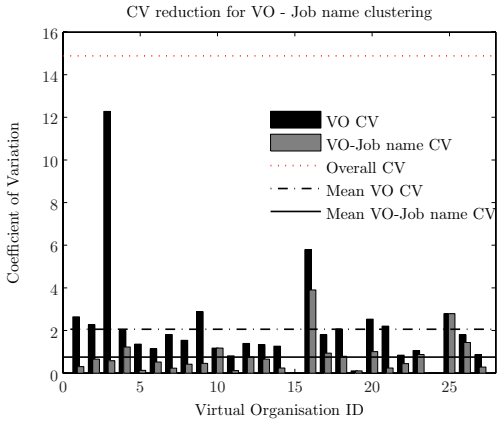


Figure 5.1: Comparison of CV values for job partitions based on owner VO with mean values of the job name clusters within each VO. Partitioning using both job properties leads to a substantial reduction in average CV values.

Submitting VO - job name set contains jobs grouped both by the submitting VO and the unique job name. The plot in Figure 5.1 shows the obvious benefit of such clustering by comparing the CV values of the overall workload, the VO set and the VO - job name set. Of the twenty seven Virtual Organisations, only two show negligible change, while all other exhibit a substantial decrease in variability. Mean coefficient of variation for this subset is 0.75 compared to 2.06 for the VO subset and 14.88 for the overall workload.

From the above presented set, a subset of 60 clusters was selected with at least 50 data points in each to enable sufficient training and validation for the forecasting algorithms. The coverage of this subset is still very high as it includes 98.26% of all submitted jobs and 65.71% of overall execution time.

Submitting VO - week number set groups jobs firstly by their submitting VO, followed by the annual week number in which they were submitted. Workload characterisation indicated that the job execution times evolve and change over time, and this set was created with the aim of capturing such behaviour. By independently treating workload generated at the different points in time, the model can develop a better fit and react faster to the fluctuations in the distribution of job execution times caused by a change in the user's scientific goal, the analysed data set or the application being used.

Figure 5.2 shows the reduction of the mean CV values for the submitting VO - week number job set compared to the submitting VO alone. The effectiveness of this compound clustering approach is clear, with all but one VO showing significant reductions in the job runtime dispersion. The mean coefficient of variation for this subset is 0.97 compared to 2.06 for the VO subset and 14.88 for the overall workload.

To ensure forecasting algorithms are only applied to clusters with a sufficient number of data points, a subset of 114 clusters from the submitting VO - week

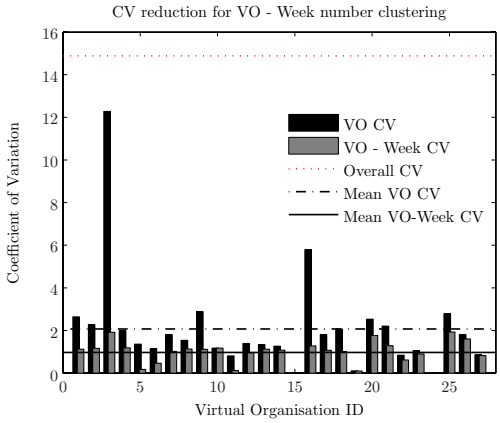


Figure 5.2: Comparison of CV values for job partitions based on owner VO with mean values of the week number clusters within each VO. Partitioning using both job properties leads to a significant reduction in average CV values.

number set with more than 100 jobs in each has been selected. The subset remains representative of the whole workload, as it covers 59.41% of overall execution time and 99.07% of all job submissions.

Submitting VO - week number - job name set is based on the successive partitioning of the entire workload based on the job's submitting VO, the week number of submission and the job's unique name. Clustering based on these three orthogonal properties produces superior results in reducing the mean variation of the data in each cluster. The purpose of this set was to test the possible increase in the predictability of the job execution times by exploiting the general execution pattern within a Virtual Organisation, the temporal locality of the job runtimes and the specific behaviour of a single application.

Figure 5.3 compares the CV values of VO, VO - week number and VO- week number - job name sets. With the mean CV of 0.59, this set is the most successful in grouping similarly behaved jobs together.

In case of this job set, its very granular partitioning of the workload created a high percentage of clusters with very few data points. For the experimental subset, only those clusters with more than 100 jobs in each week and more than 50 runs of the same job name have been selected. Due to these constraints, the resulting subset has less coverage than other sets at 56.05% of the total number of jobs included executing for 21.49% of the total runtime of the trace.

Overview of the Job Partitions

Based on the observations and the conclusions of the workload characterisation work, the whole trace was partitioned into sets using one or more of the job properties. The purpose was to reduce the variability of the job runtimes within each set, making them more predictable. This has been successfully achieved, as

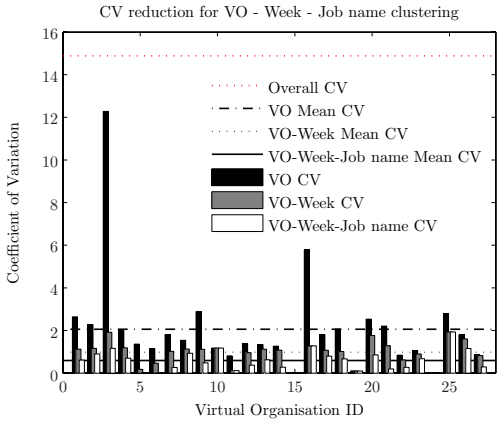


Figure 5.3: Comparison of CV values for job partitions based on owner VO and the VO-Week number with the mean CV values for the VO-Week number-Job name cluster. Partitioning using all three most important job properties leads to the lowest average CV value achieved.

demonstrated by the reduction of the coefficient of variation for each of the sets given in Table 5.1.

Set	Short Code	Mean CV
Entire workload		14.88
VO	V	2.06
Job name	J	1.62
Week number	W	6.45
VO-Job name	VJ	0.75
VO-Week number	VW	0.97
VO-Week-Job	VWJ	0.59

Table 5.1: Overview of the mean CV values of the job partitions. By comparison to the overall workload CV, a significant reduction in variability was achieved by partitioning using one, two and three job properties.

As experimental testing of different statistical forecasting algorithms requires sufficient number of data points for historical inference and subsequent statistically valid assessment of the prediction accuracy, some less populated subsets from each job partition had to be excluded. Table 5.2 summarises the coverage of the experimental workload partitions in terms of the number of job clusters, and the percentage of total job submission and total runtime those jobs attribute to.

Clearly, as clustering dimensions increase, the cluster numbers increase as well but the number of jobs within each decreases. This leads to fewer candidate groupings with sufficient number of data points and a lower coverage ratio. These subsets will be used throughout this chapter as the basis for the testing and comparison of the forecasting algorithms.

		Coverage	
Subset	No. of clusters	Job count	Run time
VO	19	99.95%	98.87%
Job name	30	97.89%	60.24%
Week number	51	100%	100%
VO-Job name	60	98.26%	65.71%
VO-Week number	114	99.07%	59.41%
VO-Week-Job	97	56.05%	21.49%

Table 5.2: Overview of experimental subsets and their properties

5.2.2 Forecasting Methods

The core assumption of this work is that, considering the properties of the Grid workload, job wallclock execution times can effectively be predicted using the time-series forecasting models. The forecasting methods chosen for the comparison reflect this assumption - the following will provide their statistical background, outline their implementation in the simulation and discuss their parametrisation.

Moving Average

One of the simplest, and certainly the most often used benchmark model, is the average or mean. While it can be applied at time t on the entire series up to $t-1$, this predictor is more often used with a sliding window averaging only the last n samples. This, "moving average" operation, which is mathematically an example of convolution, in effect smooths out the short-term variation and reveals a longer term trend. Given a time series, moving average (MA) is calculated according to the following equation:

$$\begin{aligned} F(t)_{MA} &= \frac{1}{k}\sum_{n=1}^{k}(A_{t-n}) \\ &= \frac{A_t + A_{t-1} + \cdots + A_{t-k}}{k} \end{aligned} \tag{5.2}$$

The implementation of this predictor was based on the vectorised Matlab code, and its single parameter, the size of the averaging window, was set dynamically though a feedback loop using a simple control strategy. The motivation was to reduce the time needed for the predictions to converge following an abrupt change of actual values. At each time step, the absolute percentage prediction error (see later for the definition) was compared to the accuracy of previous forecasts and used to adjust the size of the averaging window.

Related work in the predictive Grid scheduling often reports the results from a "MEAN" method whose explanation closely matches that of a moving average method. The window size ranges from a fixed value to all previous observations (implying a true mean of the whole historical series). Pro-active adjustments of the windowing has not been reported in this context before.

Moving Median

Moving median is a robust version of the moving average method. A box sliding window selects n last values of the time series and a median value is calculated as the next forecast.

This method was selected in an attempt to control the numerous outlier values present in the job execution time dataset and offer a simple, yet more robust model than the moving average. The window size for this model was adjusted using the same control procedure as for the moving average.

Simple exponential smoothing

Simple exponential smoothing (SES) could be considered as a particular type of the moving average technique, and is a prediction method often used with the financial time-series data. The forecasted value is calculated by taking a weighted average of the latest actual data and a fraction of the last predicted value:

$$\begin{aligned} F(t)_{ES} &= \alpha \cdot A(t-1) + (1-\alpha) \cdot F(t-1) \\ &= \alpha \cdot [A(t-1) + (1-\alpha) \cdot A(t-2) + (1-\alpha)^2 \cdot A(t-3) + \cdots] \end{aligned} \tag{5.3}$$

The last equation was derived by direct substitution of the defining equation into itself, and shows that as the number of past observations increases the weights assigned to the previous observations are proportional to the geometric progression $1, (1-\alpha), (1-\alpha)^2, (1-\alpha)^3, \ldots$, which is the discrete version of the exponential function after which this prediction methods was named.

The level of the smoothing is defined using the smoothing factor α; values close to unity result in less smoothing and give greater weight to the more recent observations, while values closer to zero generate more smoothed values which are less responsive to the recent changes. During the simulation, the value of the smoothing parameter was defined automatically for each job sequence through a short parameter sweep test on the training data.

Auto-regressive Method

Autoregressive (AR) approach is a commonly used method for modelling univariate* time-series. The model is a linear regression of the series against a number

*Measurements made on only one variable per observation.

of previous values of that same series. The number of historical values used for regression represents the order of the autoregressive process, which is defined by the following equation:

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t \tag{5.4}$$

where $\varphi_1, \ldots, \varphi_p$ are the parameters of the model, p is the order, ε is the error term and c the constant term defined by:

$$c = (1 - \sum_{i=1}^{p} \varphi_i)\mu \tag{5.5}$$

where μ is the process mean.

Autoregressive models are straightforward to interpret, can be fitted in deterministic time and using various methods (for Yule-Walker, Burg, Geometric Lattice and others see Chapters 17 and 19 in Pollock [188]) including the standard linear least squares techniques. The AR method used in the job execution time prediction was based on the Matlab System Identification Toolbox [189] implementation of the parameter estimation using modified covariance method. This method uses no windowing and a forward-backward approach to minimise the sum of the least squares. The requested focus of the model was set to prediction, leading to a weighting of the error function (the difference between actual and modelled values) favouring high frequencies. This minimises the one-step-ahead prediction, which typically favours fitting small time intervals.

The order of the AR model was determined automatically for each job sequence based on the partial autocorrelation (PACF) analysis of the training data. The partial autocorrelation at lag k is the autocorrelation between values of the time series at times t and $t-k$ that is not accounted for by lags 1 through $k-1$. Algorithms for computing the partial autocorrelation based on the sample autocorrelations, and the discussion of the usefulness of this method in estimating the order of the AR process is given by Box in [130] and Hannan in [190]. The orders of the AR models used in this work are selected to be the last lag on the PACF plot whose correlation value is higher than the 95% statistical significance level placed at $\pm 2/\sqrt{N}$ where N is the number of data points in the time series.

Auto-regressive Moving Average Method

Autoregressive moving average (ARMA) model is one of the most popular and effective methods for modelling time series, pioneered in the 1980s by Box and Jenkins [130]. By combining both the autoregressive and the moving average components, this model has the power to deal with random "shocks" to the

series values which propagate and influence future data points. ARMA model of AR order p and MA order q is defined by the following equation:

$$X_t = \varepsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \tag{5.6}$$

The inclusion of the moving average component complicates the fitting process as the error term (ε) is not observable. The estimation of the ARMA process parameters is therefore an iterative non-linear procedure taking a non-deterministic amount of time. The ARMA models also have a less obvious interpretation than the AR models. The implementation of the ARMA model estimation used was the one from the Matlab System Identification Toolbox based on a search algorithm minimising a robustified quadratic prediction error, with the default values for the number of maximum iterations and improvement tolerance. Further details of the algorithm are available in [189].

Estimation of the orders of the ARMA process presented the greatest obstacle in automatically applying this model. Common practice is to equate the order of the AR and MA components [191], and this was the initial assumption taken for all ARMA models used herein. The estimation of this order was the same as applied in the purely autoregressive technique. But for certain highly autocorrelated series, the AR order could be very high and, if applied as both AR and MA orders in an ARMA model, could lead to fitting problems. If these were observed, a fallback second order moving average component was used.

5.2.3 Prediction Accuracy Assessment

Different forecasting methods can be compared on a number of criteria: in specific scenarios prediction complexity or model parametrisation may be of the highest importance. Most commonly however, it is the accuracy of a model's predictions that is of primary interest. Strictly speaking, the positive or negative difference between the observed and predicted value is called a residual. The term error is often used instead, although in statistics it indicates the amount by which an observation differs from its expected value based on the whole population from which the statistical unit was chosen randomly [192]. Given this distinction, the following discussion will use the term error as it is more frequent in the subject literature.

When analysing discrete time series, calculating the spot prediction error may not be difficult, but comparing different forecast series and judging which was the most accurate may prove quite challenging. The issues of cross-series and cross-method comparison of the prediction errors have been largely neglected by the non-statisticians which tend to use inappropriate accuracy measures, mostly due to behaviour inertia. An early 1980s survey [193] found that forecasting practitioners, and academics in particular, have a strong preference for the Root

Mean Square Error (RMSE) although its pitfalls were, even at that time, already well documented. Later reviews found little has changes in last twenty years.

Past research work in the field of predictive (Grid) scheduling has reported several different accuracy measures, but few have supported their decision to use a specific measure, or discussed the implications of such decision. With no clear consensus amongst the Grid research authors on the reported accuracy metrics, direct comparison of the results of the job execution time predictions are often impossible.

The aim of this section is to properly analyse the time-series being forecasted, and select the most appropriate accuracy measure for head-to-head comparison of the forecasting methods on the same series, as well as comparison of their prediction quality amongst different job series.

The Challenges of Forecast Comparisons

The selection of an appropriate error measure depends on the nature of the data being predicted, the properties of the forecasting methods, and the objective difficulty of predicting the future series values.

Different time-series scale may cause the errors generated predicting the series with large numbers to dominate the comparison with errors obtained predicting a time-series with smaller numbers. Some of the more commonly used accuracy measures are scale-dependent, and while useful in comparing different methods on the same set of data, they should not be used when comparing the prediction errors of the data sets with different scales. Historically popular Mean Square Error (MSE), and Root MSE (RMSE), are both scale dependant and very sensitive to outlier values, leading to numerous recommendation against their use [194, 106, 195].

Since the Grid workload characterisation in Chapter 4 revealed that the job wallclock execution times are spread across eight orders of magnitude, exhibit significant long-tail behaviour, and differ substantially in the statistical location and dispersion, the use of a scale-dependant error measure for cross-series comparison would not be appropriate.

A simple way to control for the scale is to calculate the errors as the percentage of the actual predicted value. Such accuracy measures could be used to compare result across different series regardless of their scale, but have a disadvantage of being very sensitive if the actual value of the predicted data is close to zero and undefined if it is equal to zero (as it appears in the denominator of the percentage error calculation). Percentage errors also put a heavier penalty on the positive errors [196, 197], and some authors have noted their possibly skewed distribution [198].

From the aspect of the job execution time predictions, the percentage errors offer the important ability to compare the forecasting errors between the jobs

in different partitions (job sets) which would usually have significantly different scales. The execution time data also fulfils the necessary assumption of a meaningful zero required for the application of such percentage errors.

Sensitivity to the outliers is less of a problem when calibrating a prediction model, but is especially troublesome when the goal is to select the best performing prediction method. Unless those extreme values are of main interest, the errors should be trimmed to produce robust measures. To avoid an arbitrary level of trimming, and to aid direct comparison of the published results, median values are most often reported [194].

Due to the statistical properties of the job runtime sequences, and a probabilistic approach embraced in this work, a significant amount of prediction error outliers were expected. When considering the prediction errors of a single forecasting method, the main objective was to establish their central tendency. The purpose of the cross-series analysis was to analyse the increase in predictability through the use of job partitioning. Therefore, robust measures such as interquartile ranges and medians were used throughout for reporting the results across different series.

Summarising the results requires the error measure to aid in the selection of the most suitable forecasting model, and should therefore have a relationship to that decision making process. In the scenarios consisting of many different prediction methods and/or parameters, and with many numerical error measures reported, it may become increasingly hard to spot the best forecasting performer. Summary results, often trading some finer aspects of the error properties for the presentation simplicity, can be valuable in grasping the larger picture.

Direct Comparison of Forecasting Methods

The goal of the head-to-head comparison of the different prediction models was to select the best performing one for each of the individual job set. This was done by using the mean absolute error (MAE) defined as the difference between the actual and the forecasted time series values:

$$\epsilon_t = |A_t - F_t| \tag{5.7}$$

$$\mathbb{E}_{MAE} = \frac{1}{n}\sum_{t=1}^{n}(\epsilon(t)) \tag{5.8}$$

$$= \frac{1}{n}\sum_{t=1}^{n}(|A_t - F_t|) \tag{5.9}$$

where A_t is the actual and F_t the forecasted value.

The mean absolute error was selected as it is a highly sensitive measure, without outlier protection, and well suited for model calibration [194]. However being a scale-dependent metric, MAE is not intended for cross-series comparison and would be especially cumbersome to independently use on almost 400 test sequences and 5 prediction methods examined in this work.

To facilitate the comparison of the forecasting model performance, a two tier method has been used. For each job execution time sequence in each set, the mean absolute error of all the prediction models has been compared and the best one has been selected. For each of the six sets in question, a pie chart is used to depict the Percent Best [194] error statistic indicating the fraction of the set's sequences for which each of the methods has been the best performing forecaster. This avoids any bias related to the objective difficulty of predicting a certain time series, as forecast models are only compared within the same sequence. Reliable and robust [195], the Percent Best method enables direct comparison of the relative performance of each of the compared methods to all others, and a clear indication of the strength of a specific method in predicting a certain type of time series.

Cross-series Numerical Evaluation of Forecasting Errors

The Percent Best method, although valuable in judging the best prediction model, is relative and does not offer any indication of the magnitude of the forecasting errors. To assess the central tendency and the spread of the prediction errors, and in order to compare them between the different forecasting and partitioning methods, the Median Absolute Percentage Error (MdAPE) defined as follows was used:

$$\mathbb{E}_{APE} = |\frac{A_t - F_t}{A_t}| \tag{5.10}$$

$$\mathbb{E}_{MdAPE} = \text{median}(\mathbb{E}_{APE}) \tag{5.11}$$

where A_t is the actual and F_t forecasted value.

Being a percentage measure, MdAPE can readily be used to compare the error magnitudes across the series with different scales. Using the median value of APE has several benefits. It reduces the bias in favour of overestimates present in the often used Mean Absolute Percentage Error (MAPE) measure. It also makes MdAPE robust to outliers while avoiding arbitrary trimming rules, thus facilitating comparison between the reported results. It was found to have good construct validity and reliability [194], and comes well recommended for the comparison of results across a moderate number of series [195, 194, 199].

A boxplot will be used to show the location and the dispersion of the MdAPE values for each of the sequences in the job set, grouped by the prediction method.

These results are directly comparable across job sets, and indicate different accuracy levels between the forecasting methods and the job partitioning parameters.

5.2.4 Experimental Set-up

The results reported in this chapter are based on an emulated runtime forecasting system wholly implemented using the MathWorks Matlab R14 numerical analysis software. The scenario aims to replicate the job wallclock execution time prediction, the crucial step in the deadline scheduling, by presenting to the forecasting module each newly submitted job, together with its meta-data properties, and awaiting the execution time prediction. Errors between the predicted and actual values, that have occurred on the real world system, are then calculated and stored for further analysis.

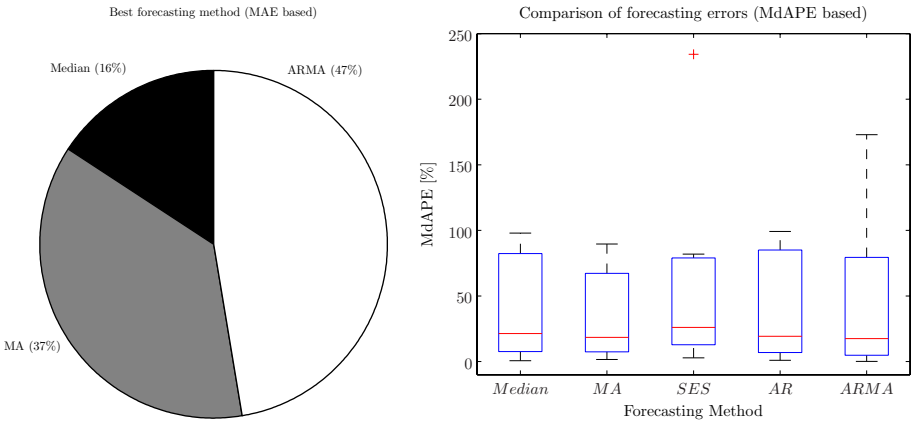
The benefit of this trace-replay system is in its use of a genuinely representative data set, which has not been modified in any aspect and thus preserves all the features and peculiarities of the real world production installation. This is an important differentiation of this work from those of fellow researchers in the field [152, 28] which have studied some aspects of the Grid workload and have decided to generate synthetic traces with characteristics similar to those they have observed.

The complete twelve months of the CCC workload was used as the basis for the experiments, sorted by the submission time, and without any data re-sampling, filtering or manipulation being done. The forecasting module was strictly ex-ante and was given access to the historical data only. No knowledge of the future was being exploited at any step in the prediction process or forecasting model parametrisation.

For the majority of the models, System Identification [189] and Statistics [200] toolboxes of the Matlab software were used. All custom prediction tools built used established forecasting formulae and were empirically validated against a well known time-series. Experiment control logic, historical trace analysis and model parametrisation heuristics were coded in Matlab and C programming languages.

5.3 Comparison of Forecasting Methods and Job Partitioning Metrics

This section will present the results of the prediction accuracy survey and offer reasons and explanations for the observed performance of the forecasting methods. The results, grouped by the job partitioning property used, are given using the Percent Best pie chart, the box plot of the MdAPE metric and its median and inter-quartile range given in a summary table.



(a) The best forecasting method (b) Comparison of forecasting errors

Figure 5.4: VO set: (a) best forecasting method (MAE based), and (b) comparison of location and dispersion of percentage prediction errors for different forecasting methods (MdAPE based). Despite the strong performance of the ARMA predictor, approaches based on sliding window (median and MA) dominate.

5.3.1 Prediction Errors: VO set

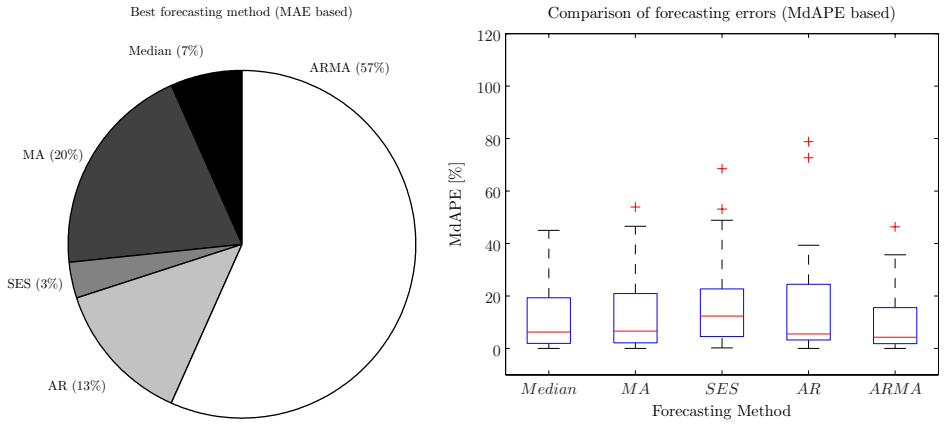
Figure 5.4 shows the performance of the different forecasting algorithms predicting the workload partitioned based on the job's submitting VO.

In almost half of the set's sequences, the ARMA method was the best predictor, followed by the moving average and the median methods. Exponential smoothing and autoregressive predictors did not score a single best forecast in this group. The poor performance of these methods can be blamed on the evident short-range dependence of job runtimes in this set which suits the sliding window predictors better. The ARMA model excelled in predicting this job set mostly due to the automatic parametrisation method which has repeatedly chosen high orders of the moving average process.

Table 5.3 summarised the prediction errors using the MdAPE metric. The median values are in the 17.5% - 26.1% range, and few outliers are present. The dispersion of the errors is high however, mostly caused by the variability of the job set and its relatively high CV value.

	Forecasting Method				
	Median	**MA**	**SES**	**AR**	**ARMA**
25th Percentile	7.62	7.43	12.83	6.89	4.82
Median	21.32	18.46	26.09	19.30	17.54
75th Percentile	82.37	67.20	79.04	85.07	79.44

Table 5.3: VO set: Comparison of location and dispersion of prediction errors (MdAPE based) for different forecasting methods



(a) The best forecasting method (b) Comparison of forecasting errors

Figure 5.5: Job name set: (a) best forecasting method (MAE based), and (b) comparison of location and dispersion of percentage prediction errors for different forecasting methods (MdAPE based). Strong combined success of auto-regressive predictors (AR and ARMA) indicate that successive runtimes of individual jobs are highly autocorrelated.

5.3.2 Prediction Errors: Job name set

The best forecasting methods, and the distribution of the prediction errors for the Job name set are shown in Figure 5.5. The ARMA method delivers the lowest error forecasts in almost 60% of the series in this set, followed by MA and AR methods. The boxplot reveals that the predictions for this set are much more accurate than those for the VO set, with median MdAPE ranging from 4.3% to 12.3%. The dispersion of the MdAPE values is much smaller, and even with a few outliers the top quartile for the ARMA method is only 20%. These values are summarised in Table 5.4.

Good performance of the ARMA and AR methods on this set indicates that run time sequences of individual jobs are highly autocorrelated and can be used to produce good quality predictions. While tracking job names or applications may not be easy using the current Grid middleware, for all the reasons previously identified in Section 2.1.3, the benefit of this information to the predictive scheduling is certainly a strong motivation for the better integration of the application identity into the Grid monitoring and workflow management components.

5.3.3 Prediction Errors: Week number set

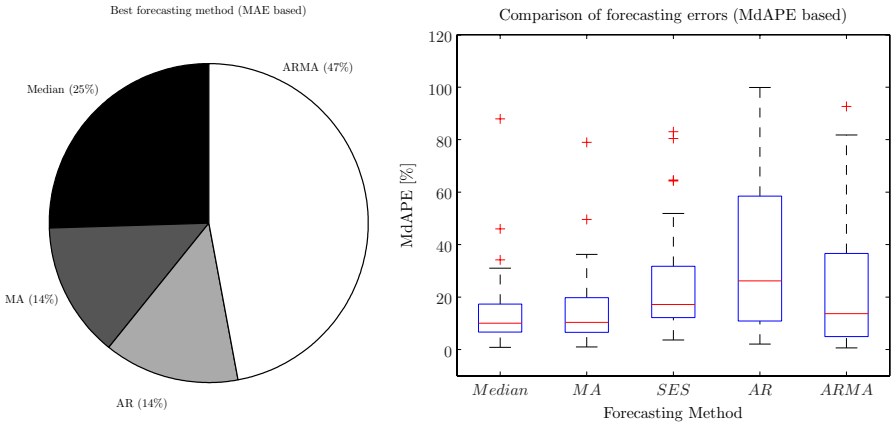
The results of the job runtime predictions for sequences from the Week number set are given in Figure 5.6. The Percent Best pie chart shows ARMA method leading other methods in the forecasting accuracy, followed by the Median, MA and AR methods. The notable performance of the Median predictor is understandable considering the very high coefficient of variation of this set, and the lack of separation of the user groups and jobs with different statistical properties within

	Forecasting Method				
	Median	**MA**	**SES**	**AR**	**ARMA**
25th Percentile	1.92	2.17	4.55	3.26	1.83
Median	6.25	6.64	12.32	5.54	4.28
75th Percentile	19.33	20.94	22.71	24.48	15.56

Table 5.4: Job name set: Comparison of location and dispersion of prediction errors (MdAPE based) for different forecasting methods

it. As there is very little autocorrelation of the successive job runtimes in this set, a robust average produces competitive results. The MdAPE boxplot further shows that Median and Moving Average errors were less dispersed that those of other forecasting methods.

A summary of the MdAPE statistical properties is given in the Table 5.5. The Percent Best and MdAPE error statistics may seem at odds here, since the best performing algorithm according to the Percent Best method does not have the lowest MdAPE median value. However if one considers that the Median and ARMA methods perform at their best in predicting very dissimilar series, it is entirely possible for one method to be better at a large number of individual sequences, and perform so poorly at a number of others as to raise its median error considerably. Whiskers on the ARMA boxplot further confirm this was the case.



(a) The best forecasting method (b) Comparison of forecasting errors

Figure 5.6: Week number set: (a) best forecasting method (MAE based), and (b) comparison of location and dispersion of percentage prediction errors for different forecasting methods (MdAPE based). Median predictor performs well due to the high CV value of this set and lack of job separation based on their statistical properties.

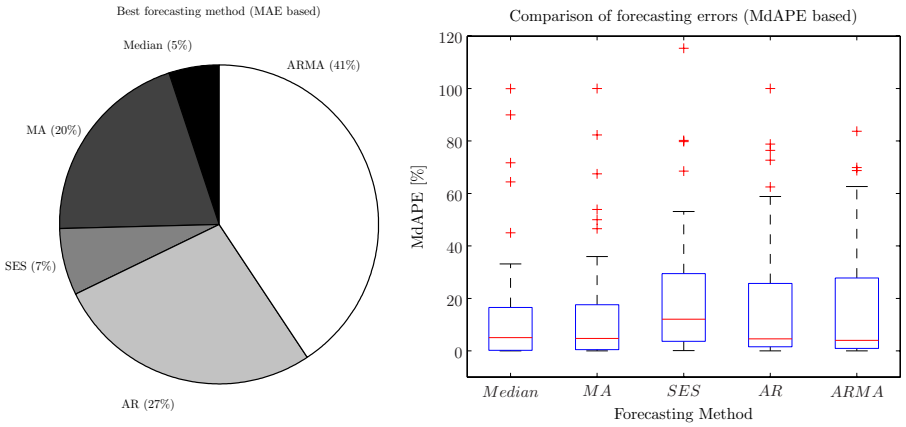
	Forecasting Method				
	Median	**MA**	**SES**	**AR**	**ARMA**
25th Percentile	6.69	6.59	12.20	10.91	4.95
Median	10.06	10.34	17.18	26.20	13.68
75th Percentile	17.32	19.76	31.75	58.47	36.65

Table 5.5: Week number set: Comparison of location and dispersion of prediction errors (MdAPE based) for different forecasting methods

5.3.4 Prediction Errors: VO - Job name set

Figure 5.7 shows the forecasting results of the first multiple metric set, the VO - Job name set. While still delivering the highest percentage of the best predictions, the ARMA method is less dominant, and is closely followed by the AR and MA methods. Evidently, job partitioning according to both the originating Virtual Organisation and the job name sufficiently isolates execution patterns for the time-series forecasting methods based on autocorrelation properties to perform best.

Although the boxplot, and the summary data in Table 5.6, reveals a larger inter-quartile range of the MdAPE values for the AR and ARMA methods, Median and MA methods have a significantly larger number of outliers. The lower quartile of the errors is very low for all prediction methods, further confirming that predictions of execution times for this group are of very high quality.



(a) The best forecasting method (b) Comparison of forecasting errors

Figure 5.7: VO-Job name set: (a) best forecasting method (MAE based), and (b) comparison of location and dispersion of percentage prediction errors for different forecasting methods (MdAPE based). Autoregressive methods perform well and suffer from less extreme outlier error values despite a larger interquartile range

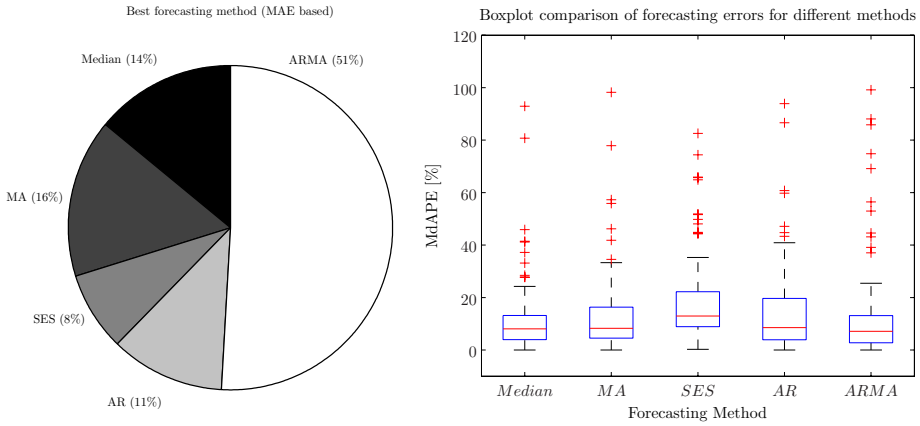
	Forecasting Method				
	Median	**MA**	**SES**	**AR**	**ARMA**
25th Percentile	0.24	0.51	3.65	1.59	0.94
Median	5.08	4.76	12.07	4.61	4.04
75th Percentile	16.56	17.57	29.47	25.72	27.82

Table 5.6: VO-Job name set: Comparison of location and dispersion of prediction errors (MdAPE based) for different forecasting methods

5.3.5 Prediction Errors: VO - Week number set

The performance of the job execution time prediction of the different forecasting methods on the VO - Week number set is given in Figure 5.8. The ARMA method achieves highest Percent Best score, followed by the MA, AR and Median approaches. The boxplot indicates outliers are present with all prediction methods, but the inter-quartile range of the error values is small, especially so in the case of Median, MA and ARMA predictors.

Median and quartile values of the MdAPE metric, given in Table 5.7, show a significant increase in the prediction accuracy compared to the results of the VO set. Clearly, the addition of a temporal dimension into the workload partitioning has managed to better group similar job runs, and has therefore led to an increase in the prediction accuracy of the execution times.



(a) The best forecasting method (b) Comparison of forecasting errors

Figure 5.8: VO-Week number set: (a) best forecasting method (MAE based), and (b) comparison of location and dispersion of percentage prediction errors for different forecasting methods (MdAPE based). ARMA predictor performs best with fewest extreme outlier error values and second smallest interquartile range.

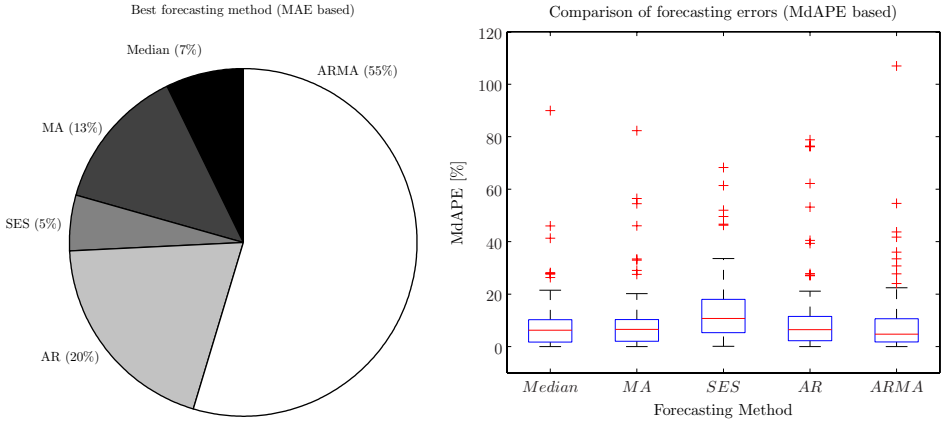
	Forecasting Method				
	Median	**MA**	**SES**	**AR**	**ARMA**
25th Percentile	3.98	4.55	8.90	3.92	2.75
Median	8.07	8.29	12.99	8.55	7.13
75th Percentile	13.18	16.37	22.22	19.68	13.13

Table 5.7: VO-Week number set: Comparison of location and dispersion of prediction errors (MdAPE based) for different forecasting methods

5.3.6 Prediction Errors: VO - Week number - Job name set

The prediction results for the VO -Week number - Job name set are shown in Figure 5.9. Almost three quarters of the sequences in this set were best predicted using either the ARMA or AR methods, with the ARMA proving best in 55% of the cases. Again, such high success rate of these methods indicates a highly autocorrelated time-series with lower levels of variability. The boxplot shows the distribution of the MdAPE metric with some outliers, but with a very low dispersion of error values.

Summary data given in Table 5.8 confirms that the prediction errors achieved in this set are superior compared to all other job partitioning sets. The ARMA forecasting method managed to predict the execution times with the median MdAPE value of only 4.75% and the upper quartile value of only 10.61%. Such results confirm the added value of the multi-dimensional partitioning of the workload using job meta and temporal properties. The resulting job partitions lock onto the underlying workload patterns, thus reducing execution time variability



(a) The best forecasting method (b) Comparison of forecasting errors

Figure 5.9: VO-Week number-Job name set: (a) best forecasting method (MAE based), and (b) comparison of location and dispersion of percentage prediction errors for different forecasting methods (MdAPE based). This multidimensional job partitioning is best predicted using the ARMA method which delivers lowest median MdAPE error of all job sets.

	Forecasting Method				
	Median	**MA**	**SES**	**AR**	**ARMA**
25th Percentile	1.75	2.06	5.32	2.24	1.80
Median	6.27	6.57	10.71	6.46	4.75
75th Percentile	10.28	10.29	18.02	11.49	10.61

Table 5.8: VO-Week number-Job name set: Comparison of location and dispersion of prediction errors (MdAPE based) for different forecasting methods

and making them more predictable.

5.4 Chapter Summary

Considering the amount of comparative data presented, the chapter will conclude with an overview of the experimental results. The summary will address two main aspects of the work separately: the performance of the forecasting algorithms and the benefits of job partitioning.

5.4.1 The value of prediction methods

Considering the computational and implementation expense of the advanced time-series forecasting algorithms, the natural question is to ask whether they indeed provide an increased prediction accuracy in the job execution time prediction scenario. The Percent Best method again provides a valuable overall comparison between forecasting models based on the highly sensitive mean absolute error (MAE) metric.

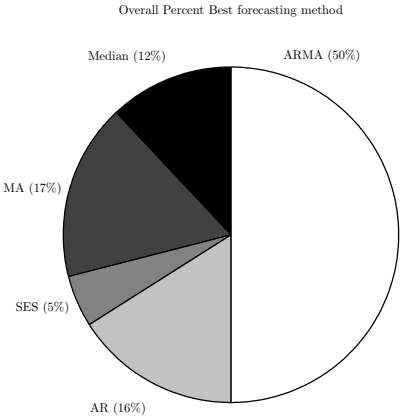


Figure 5.10: Comparison of overall performance of prediction methods across all job sets for different prediction methods using Percent Best statistic (MAE based). The most sophisticated ARMA method has performed better in more sets than all other predictors put together.

	Median	MA	SES	AR	ARMA
25th Percentile	13.24	15.38	19.45	36.01	13.21
Median	41.38	47.18	63.83	68.94	49.18
75th Percentile	117.10	140.23	206.98	169.63	158.92

Table 5.9: Comparison of prediction error (APE based) for different forecasting methods applied to non-partitioned workload. Results are considered as a benchmark for judging the benefits of workload partitioning using different job properties.

Figure 5.10 shows a summarised Percent Best statistic for every job sequence in every job set presented in this chapter. The exponential smoothing method is the overall worst performer and has been the best predictor in only 5% of the job sequences. The Median predictor, with 12% of the lowest mean absolute prediction errors, is a simple to implement, computationally inexpensive alternative for forecasting an occasional job execution time series with a very poor autocorrelation and a high degree of variability.

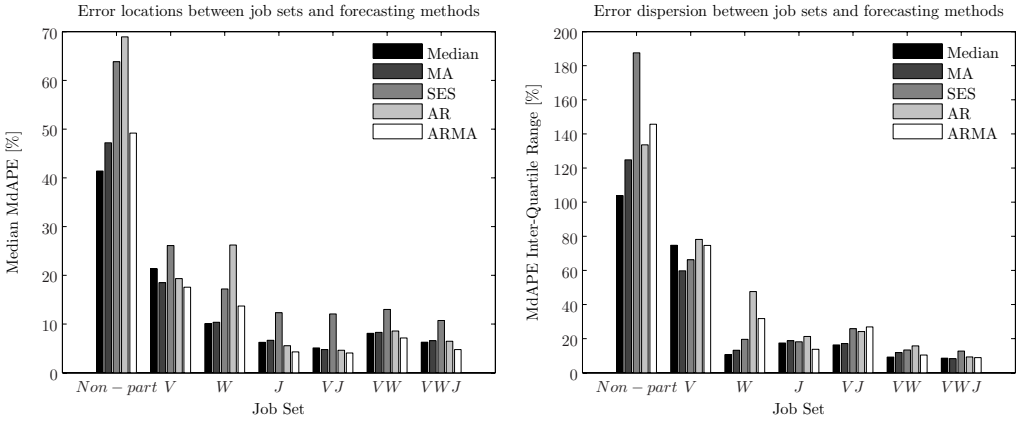
The combined performance of the AR, MA and the ARMA predictors returns lowest error forecasts in the overwhelming 83% of all job sequences. Considering the ability of the ARMA model to behave as a purely autoregressive or purely moving average predictor (by setting the order of the AR or MA component to zero), a generalised implementation with a suitable parametrisation technique would provide superior performance in predicting the job execution times characterised by a wide range of statistical properties.

5.4.2 The value of job partitioning

The experiment showed a significant and sustained increase in the prediction accuracy of all forecast methods as jobs were partitioned into clusters with an increasingly more consistent behaviour. To establish a benchmark against which this added accuracy could be judged, all five predictors were run on the whole year long trace without applying any partitioning criteria. Table 5.9 gives a summary of the absolute percentage forecasting error location and dispersion for this non-partitioned workload.

To summarise the findings, Figure 5.11 gives a side by side comparison of the medians (a) and the inter-quartile ranges (b) for all five forecasting methods and all job partitioning approaches including the non-partitioned benchmark.

The plots show an obvious reduction in the median absolute percentage error for all prediction models as the workload is partitioned using an increased number of job properties. The lowest median error in the set partitioned using three orthogonal job properties is almost ten times smaller than the lowest median error in the non-partitioned workload. The addition of the temporal property based on the job's submission time has a noted positive effect.



(a) Location (MdAPE median) (b) Dispersion (MdAPE interquartile range)

Figure 5.11: The benefit of multidimensional job partitioning is clearly shown by comparing the location (a) and dispersion (b) of MdAPE error values for non-partitioned and clustered job sets.

Job partitioning has also decreased the dispersion of the prediction errors in all job sets as compared to the non-partitioned job sequence. The implication of this effect is that measures of the error sample central tendency, such as the median, are more representative of the distribution's real statistical location. This is at least as important as the median accuracy, as the prediction error is more bounded.

Chapter 6

Deadline Scheduling Evaluation

Having thoroughly analysed a representative production Grid workflow and devised methods for predicting the job execution times, the focus in this chapter will be on demonstrating the usability of such forecasts in delivering deadline scheduling on the Grid.

The following sections will present the purpose-built scheduling simulator and a predictive scheduling algorithm not previously used in the Grid context. The improvement in deadline adherence of the predictive algorithm will be compared to the commonly used FIFO queue. The simulation results comprise two different deadline generation algorithms and two job execution time forecasting methods. They demonstrate the value of the predictive scheduling approach and the importance of prediction accuracy.

6.1 Motivation and Scope

The forecasting framework presented in the previous chapter enables the scheduler to independently estimate the runtime of jobs waiting in the queue - a highly desirable functionality which can aid in many aspects of the scheduling including the widely used backfilling* technique [201], and yield management approaches to maximising service cluster profitability (see Appendix C.7). However, the primary motivation behind the simulation effort in this chapter is in establishing whether, and by what amount, the forecasted job execution times can help the scheduler turn the workload around to a certain, user requested, deadline.

To that end, a new scheduling algorithm understanding the notion of the job deadline and able to make use of the predicted job runtimes was needed. A worthy candidate was found in the real-time systems domain, and was for the first time applied to a job scheduling problem in the Grid context. Most importantly, no

*The optimisation process queueing smaller and shorter jobs ahead of the larger ones which are unable to start due to insufficient resources.

production or experimental distributed platform collects or stores the historical data on the requested turnaround times or job deadlines. As the performance of any deadline scheduler is highly influenced on the distribution of deadline times, a sensible generation model rooted in the empirical observations had to be selected.

With these goals in mind, the simulation runs were structured to answer the following three questions:

1. Can a predictive scheduling method deliver better job deadline adherence than the currently used batch approaches?

2. Considering the lack of the data on the user requested deadlines, how sensitive would the performance of the predictive scheduling be to different deadline generation models?

3. Does the improvement in the job runtime forecasts translate into better deadline adherence or not?

The above questions have focused the simulation implementation and indicated important limitations to its scope. Scheduling of distributed and parallel workloads is an extensively researched topic grounded in the statistics and optimisation techniques. It was not within the scope of this work to propose, or indeed compare, different predictive scheduling techniques, some of which were previously discussed in the literature survey chapter (see Section 3.1.2). Algorithms making better use of the job runtime predictions may exist, or be in development. The main aim of this simulation was to empirically show that the job runtime forecasts, of quality attained by the methods presented in this thesis, coupled with a reasonable predictive scheduling technique can lead to deadline scheduling with better deadline adherence than it is currently possible with the first-come-first-served methods.

Although two very different algorithms for generation of job deadlines have been tested, until the actual data from the first production deadline scheduler is available it is not possible to be certain of the distribution deadline values will have, their correlation with the actual job runtimes, or with other social and economical aspects.

6.2 Specific Methodology

The development of the scheduling simulator was supported by a specific methodology in the choice of the software coding language and technique, generation of the job deadlines, implementation of the novel scheduling method, and the selection of the performance metrics on which the new approach will be judged.

6.2.1 Scheduling Methods

The selection of the scheduling methods to be used was influenced by the purpose of the simulation: to test whether the job runtime predictions generated by the developed forecasting method can deliver job scheduling to a user requested deadline.

The benchmark scheduling method, still in very wide use in the production Grid clusters, is the basic FIFO queue, or first-come-first-served (FCFS) scheduling. UCL's CCC Grid facility, from which the original workload trace was sourced, also uses this scheduling method. FCFS scheduling is implemented in the simulation by maintaining a stack of jobs in the order in which they were submitted. New jobs are appended at the tail of the stack while available nodes are sent jobs from the stack's head.

With the availability of job runtime predictions, a deadline scheduling method is able to calculate the latest possible job start time in order to still make the requested deadline. By delaying the execution of the job until the remaining deadline time is just enough to finish the job, the resources are kept available in case a job with a "tighter" deadline arrives. This approach is the deadline scheduling method of choice in this simulation, and will be referred to as latest time to run first (abbreviated LTTR). It was implemented in the simulator by calculating the latest required job start time, as a difference between the job requested deadline and the predicted job run time, and sorting the entire queue in the ascending order:

$$LTTR(i) = t(i)_{deadline} - t(i)_{estimate} \tag{6.1}$$

The inspiration for implementing the LTTR deadline scheduling was drawn from the extensive research in the scheduling of the real-time systems using the earliest deadline first (EDF) algorithm. For a system with n independent tasks, all ready at time $t = 0$, where each job J_i has a deadline d_i, the lateness of a job i is defined as $l_i = f_i - d_i$, where f_i is its completion time [202]. The maximum lateness of all jobs, provided the schedule in non-preemptive, can then be minimised by an earliest deadline first algorithm which places the jobs in the order of non-decreasing deadlines. This algorithm was originally given by Jackson in 1955 and has proven to be optimal in [203]. If the scheduling problem is altered so that not all jobs are released (submitted) at time $t = 0$ the scheduling problem becomes NP-hard, as shown by Graham and Lenstra in [204]. Allowing preemption generally makes the scheduling process easier, and Liu and Layland have in [205] proved the optimality of the EDF algorithm for such schedules.

LTTR is in essence an earliest deadline first approach, although the deadline (in the real-time systems sense) which the algorithm optimises on is not the actual user requested deadline, but rather the computed latest time at which the job could begin running and still finish within the limits of the user's requested

turnaround time. This important distinction allows for the non-correlated nature of the actual job runtimes and user's deadlines, enabling the system to make a decision on what the real deadline for starting the job is. The application of the EDF approach to the Grid scheduling has been possible because the execution time of each queued job can be predicted using the forecasting engine presented in this thesis.

6.2.2 Scheduling Simulator

Due to the fact that a real Grid trace was used, the scheduling simulator was in effect a trace replay system. Since the workload consisted of independent, sequential tasks, the effect a different scheduling strategy would have on deadline adherence could simply be observed by changing the way in which queued jobs are dispatched. Hence, the scheduling simulator was expected to execute the following tasks in an efficient way:

1. Queue the incoming jobs for execution in a specific order stipulated by the scheduling method being examined.

2. Obtain the runtime prediction for each submitted job from the forecasting subsystem.

3. Simulate the assignment of jobs to a number of work nodes in a master-slave fashion.

4. Following the execution period equal to the actual job runtime on the real cluster record whether the deadline was missed and by what amount.

The simulator was implemented in ANSI C with API calls to MATLAB in order to interface with the job runtime prediction engine. Figure 6.1 shows the programme structure of the simulator.

The first stage of the simulator is the initialisation of the data structures and the parsing of input parameters such as the number of worker nodes and the starting time of the simulation. The waiting queue and the list of free worker nodes are implemented as singly linked lists and these are also created at this stage.

The main programme loop is the simulation clock, of which each increment corresponds to one second - the sampling period of the accounting data collected from the production cluster. The loop is entered until no more jobs are available in the input file, no jobs are waiting in the queue and all nodes have finished running jobs assigned to them. The simulator thus ensures all jobs submitted in a given workload trace are run to completion and their deadline statistics captured.

At each time increment, the incoming job queue is checked for newly submitted jobs and these are placed in the queue. Depending on the scheduling method

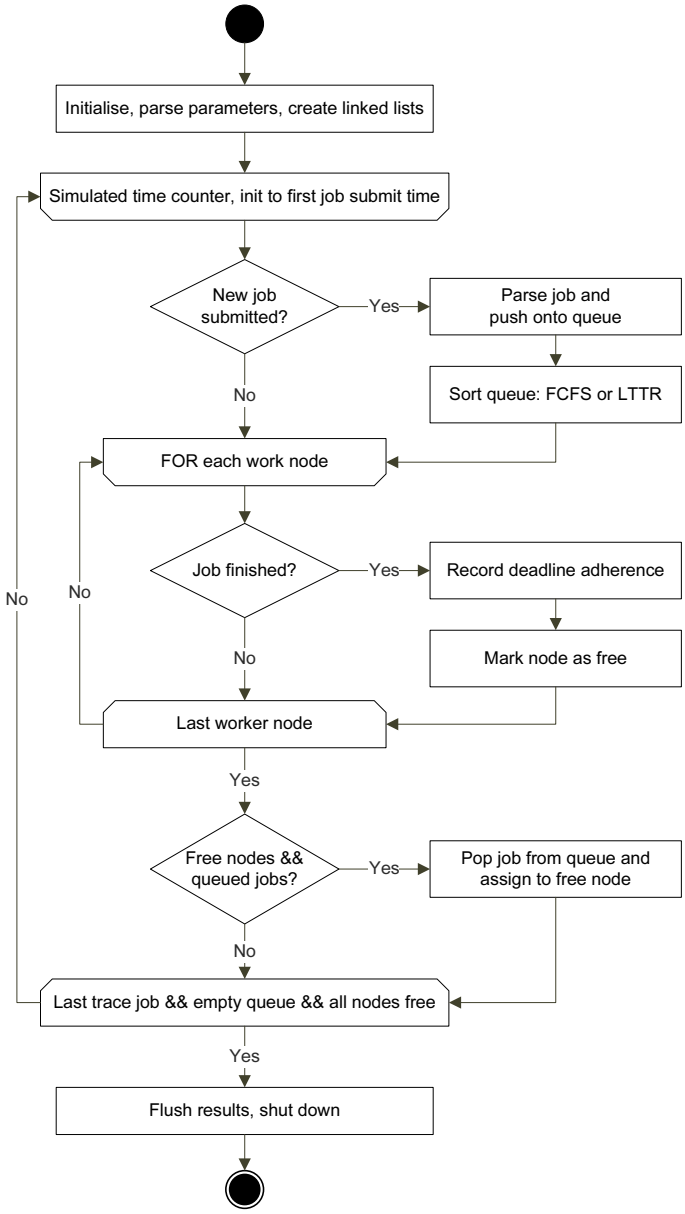


Figure 6.1: Flowchart diagram of the scheduling simulator implementation

being simulated, the queue is then either kept sorted by the job submission time (FCFS), or re-sorted by the latest time to run (LTTR). The sorting of the linked list entries in this step is the most time consuming part of the simulation, and various optimisations were applied to increase the speed of this operation.

The simulator then proceeds to check each of the worker nodes for completed jobs, *i.e.* those jobs whose end time is equal to the current clock time. The number of the worker nodes in a simulation run is arbitrary but constant, and for all the results reported in this section was set to 100 to match the number of nodes of the actual CCC Grid from which the workload was sourced. For each completed job found, the simulator records the amount of time (in seconds and as a percentage of the actual job runtime) the job has underrun or overrun the

deadline, and moves the node to the free node list.

The final stage within the main programme loop is the assignment of the waiting jobs to the available worker nodes. All worker nodes are treated equally and the jobs are presumed to have no dependency between them. Finally, if the conditions for exiting the main time loop have all been met, the simulator proceeds to flushing all file streams, releasing memory and shutting down.

The simulator was compiled using GCC ver. 4.1.1 under CentOS 5 running on Sun hardware. Each simulation run compromising the whole year's worth of the workload trace took around 12 hours to complete.

6.2.3 Workload Trace

The simulation was entirely driven in a trace-replay fashion by the real workload collected on the production system: at no point does any part of the simulator see into the future nor makes any use of the events that have not yet occurred within the simulated time. The advantages of using the real workload trace are in its authenticity and heterogeneity, which may cause some difficulty explaining the simulation behaviour. Synthetic traces, which can easily be parametrised and sized, are always dependent on the quality, and the assumption made, by the generation algorithm. With this in mind, the production trace was selected for this simulation as it was extensively studied and had its representativeness confirmed in Chapter 4.

The simulation trace spanned the full 12 months of the period in which the CCC Grid cluster was monitored. The job runtime forecasting subsystem used the three dimensional partitioning based on the owner VO, week of submission and the job name, as introduced in Section 5.3.6. This job partitioning set was selected due to its superior prediction performance and the use of both the job's temporal and meta-data. To study the effect of the quality of the runtime predictions on the deadline adherence, two forecasting methods were compared. The simple median predictor (see Section 5.2.2) was contrasted to the best performing ARMA predictor (see Section 5.2.2). These results will be reported as LTTR-MD and LTTR-ARMA respectively.

6.2.4 Job Deadline Generation

Job deadlines are a novel metric in the context of job scheduling on the Grid and as such have not been used in the production systems or recorded in the existing workload traces. However, many of the backfilling job schedulers that require some indication of the job execution time have required that users state anticipated runtime of their jobs, and have made this information available through the accounting logs. This information was extensively studied and ways of modelling the user estimates have previously been suggested by Mu'alem [206], Tsafrir [207], Feitelson [208], Cirne and Berman [23] and others [59, 116].

The author of this thesis proposes that the user estimates of job runtimes can be used as the basis for the generation of the missing job requested deadlines. Both metrics are user submitted time values and bear some relation to the amount of time they estimate (or would like) their jobs to run for. In the present schedulers user runtime estimates are treated as maximum execution time values, and jobs are killed upon reaching these times. The user is therefore inclined to grossly overestimate: research has shown that the maximum allowed runtime is the most often supplied user estimate [207, 116]. This would probably not hold true in the case of a simplistic user requested deadline, where the tendency would certainly be to request the shortest possible turnaround. However, coupled with a Grid economy system, the users could be given a strong incentive to specify the latest time after which the results of the job would have no value to them, thus increasing the proportion of relaxed deadlines and bringing the statistics closer in line with that of user runtime estimates. For simulation purposes, the deadlines were created using two different user runtime estimate modelling algorithms that have been commonly used in the literature.

Uniform Job Deadlines

With the uniform distribution deadline approach, the actual runtime of each job is multiplied by a random number drawn from a uniform probability distribution and added to the job's submission time to generate the requested deadline:

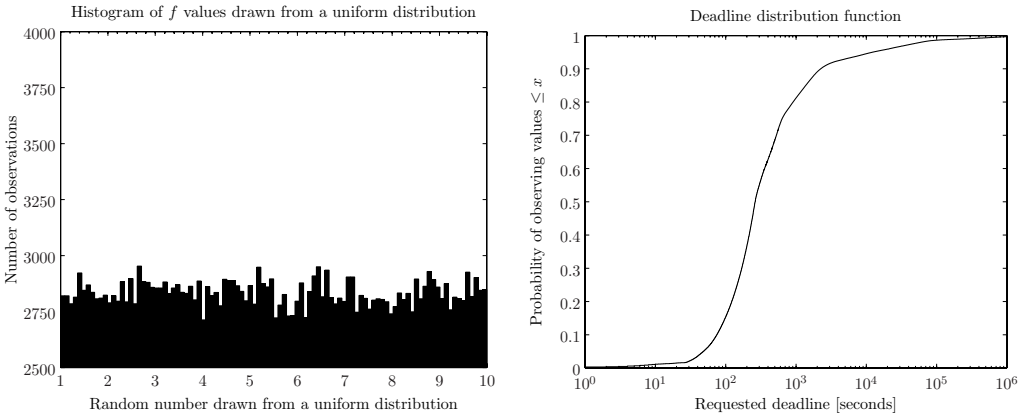
$$D(i) = t(i)_{sub} + (rt(i)_{act} * f(i)) \tag{6.2}$$

This model, proposed in [206] and used in [209, 210], is also known as the "f-model" as it assumes that the job runtime estimates are uniformly distributed within $[rt, (f+1)rt]$ where rt is the job runtime and f is some non-negative factor. Clearly, f values of less than one generate unfeasible deadlines and, although these are likely to occur in the real world, are not used in this simulation. It is therefore a common practice to draw the deadline multiplier values from a distribution between 1 and 10, 20, 50, 100 or even 300. To create a challenging environment, a very low multiplier of 10 was selected for the deadlines used in the simulation. Therefore, no deadline was longer than ten times the actual execution time of the submitted job.

The histogram of the typical values drawn for the runtime multiplier, and the resulting distribution of the deadline times are shown in Figure 6.2. The distribution of the deadlines closely resembles that of the job runtimes, shown previously in Figure 4.15, as these are simply related by the f multiplier.

Modal Job Deadlines

As an alternative, Tsafrir [207] has suggested, based an extensive research, that a more realistic model of the user runtime estimates, and therefore requested



(a) Histogram of deadline multiplier values (b) Deadlines distribution function

Figure 6.2: Histogram of deadline multiplier values f drawn from a uniform distribution between 1 and 10 and the corresponding requested deadline cumulative distribution function.

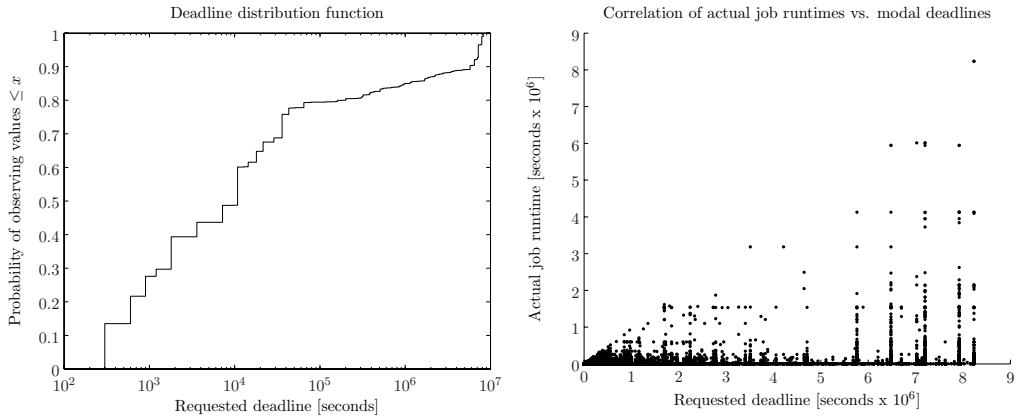
deadlines, would be highly modal. Humans have a known tendency to round up time to convenient values such as 5, 15, 30, 60 minutes and 1, 2, 6, 12, 24 hours. By analysing the available workload traces containing user runtime estimates, Tsafrir has developed a methodology and tools for generating realistic estimate values. Modal deadlines used in this scheduling simulation are based on theses findings and have been generated using a modelling toolbox developed by Tsafrir [211].

The notable departure from the model was is the specifying the fraction of the jobs that were assigned the highest deadline value, that equal to the longest running job in the trace (in the CCC example this is around 3 months or close to $8 \cdot 10^6$ seconds). Tsafrir and others have found that this value often attributes to almost a quarter of all user runtime estimates, but for this scheduling simulation this fraction was reduced to just 1% creating a very demanding deadline profile.

The resulting deadline distribution is shown in Figure 6.3(a), its step-like shape indicating strong modality and the preference for human-favoured values. Following Feitelson's findings, the scatter plot of the job actual runtimes and their corresponding deadlines, Figure 6.3(b), shows a very weak correlation between the two. This would certainly hold for the job deadlines as well: provided all deadlines are feasible, their duration would only be conditional on the urgency of the job and its value to the user, and not on its actual execution time.

6.2.5 Performance Metrics

Judging the performance of the scheduling method, and the impact different job runtime prediction approaches have on the deadline adherence, becomes a challenging task when a long, highly heterogeneous, production workload trace is used. The simulation results will therefore be assessed on the following three



(a) Deadline distribution function (b) Runtime vs. Deadline scatter plot

Figure 6.3: Cumulative distribution function of deadline multiplier values f generated using a modal algorithm and showing strong preference for human "round" values. The scatter plot indicates a very weak correlation between job runtime and requested deadline.

metrics:

Deadline hit ratio is the first and the most obvious performance metric. The ultimate goal is to maximise the number of jobs finished before their deadline for any given workload. While this criteria is easy to relate to, it treats all missed deadlines equally, without respect to the amount of the deadline overrun. In a soft-deadline system, such as the proposed Grid deadline scheduling, a certain degree of leniency is implied and a small amount of deadline overrun may be acceptable (provided a certain virtual "monetary" credit is given back to the user in the Grid economy concept).

Deadline breakage statistic looks at the location and dispersion of overrun times and tries to explain in more detail how well the scheduler has managed the deadlines. Clearly, an approach with a lower average overrun, lower dispersion and a smaller number of outlier values is more desirable and leads to better, more dependable performance. In examining the amount of deadline breakage both absolute (seconds) and relative (percentage of the actual job runtime) values will be considered. Any scheduler bias, or preferential treatment of a certain class of jobs, would be made obvious by a larger disagreement of these two measures.

Underrun and overrun distributions plotted as the cumulative distribution functions round up the analysis of deadline adherence for each scheduling method and offer a way of direct comparison. Preferably, overrun times distribution should be head heavy, and can be used to study the effect that "softening" the deadlines would have on the fraction of completed jobs. Distribution of deadline underruns is equally important, as heavy tail behaviour indicates lower optimisation with

more slack time and thus lower overall utilisation. Ideally, the scheduler would have all the jobs finish as close to the deadline as possible to increase the chance of servicing an unexpected demand of short deadline (and thus high value) jobs.

6.3 Deadline Scheduling Performance

This section will present and discuss the results of the scheduling simulation using different job execution time forecasting algorithms and scheduling methods. Care was taken to present the same input workload to the simulator on each run, and in cases where this was not strictly possible (for example due to different training requirements of the MD and ARMA prediction methods the number of jobs was slightly different), checks were made to ensure the overall integrity of the workload.

Given the importance of the deadline distributions on the scheduling performance, results are reported separately for the two deadline generation methods.

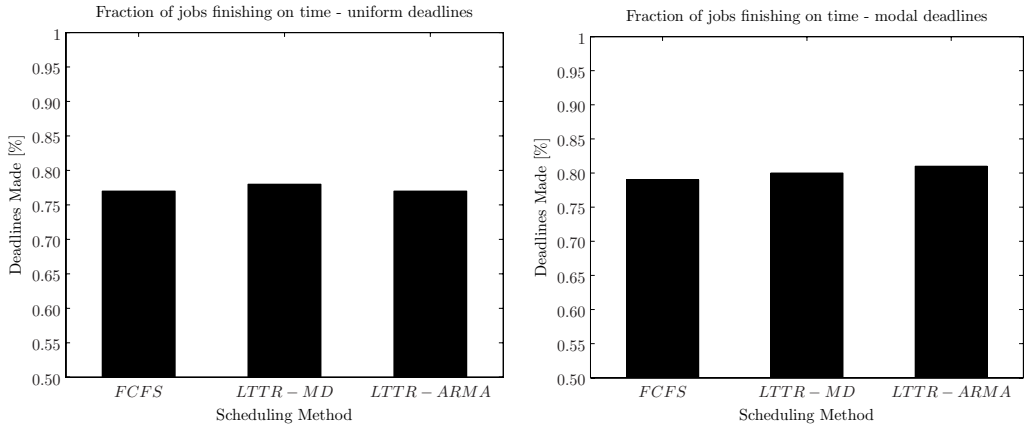
6.3.1 Fraction of Deadlines Made

Two bar chart plots in Figure 6.4 show the percentage of jobs that have been run and completed prior to their, simulated, user requested deadline. Immediately obvious is the fact that at least three quarters of the jobs, whether scheduled using FCFS or LTTR strategies, finished before the deadline. The results also reveal a rather small difference between the on-time completion of the jobs scheduled using the FCFS and the predictive scheduling methods. For uniform deadline distribution, the best performing method is the LTTR-MD followed by the LTTR-ARMA. The difference between each of these and FCFS is around 1% - 1.5% or 6000 to 7500 jobs. In the case of modal deadlines, LTTR-ARMA is clearly the best performing method leading FCFS by almost 5% (or 30,000 jobs).

The deadline hit ratio metric, although showing a measurable level of performance improvement, suggested that the benefit of using the predictive scheduling method was less than anticipated. In depth analysis of the job arrivals and their durations in the input workload revealed that for the first six months the facility was able to service all the submitted jobs with a manageable amount of contention. Around week 34 however, the cluster had suddenly become saturated with numerous submissions of very long running jobs (see the “hotspot” in Figure 4.40 on page 106). This causes all the jobs submitted after this time to miss their deadlines due to the lack of available resources, regardless of the scheduling methods applied.

6.3.2 Deadlines Breakage Statistics

The starvation of resources that the Grid was experiencing further stresses the need to compare the amount of deadline overrun between the scheduling methods,



(a) Uniform deadlines (b) Modal deadlines

Figure 6.4: Percentage of jobs finishing on or before their requested deadline for uniform and modal deadline distributions. For uniform deadlines, predictive methods achieve around 1.5% improvement, while for modal deadlines the adherence is increased by almost 5%

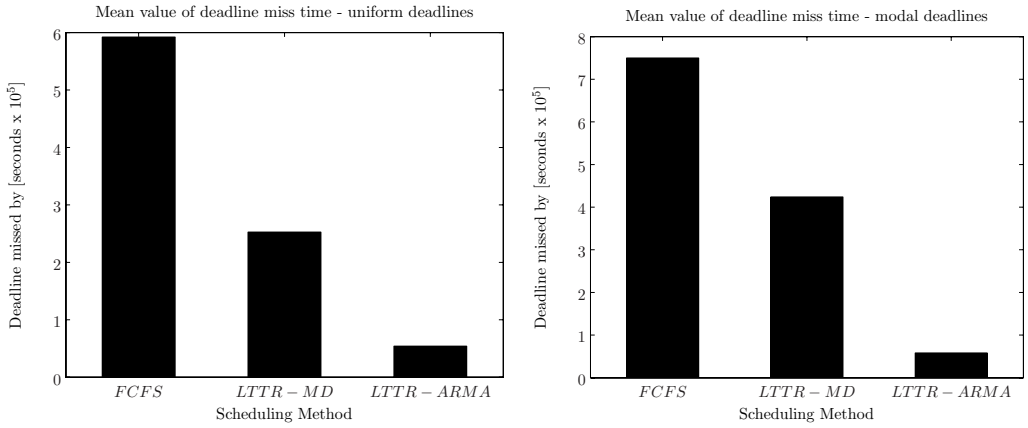
and use it to assess which approach has managed to best minimise the negative effect of the lack of resources.

Comparison of Breakage Times between Scheduling Methods

Figure 6.5 compares the mean deadline miss times (in seconds) between the scheduling methods for both uniform and modal deadlines. The benefit of the predictive approach in now clearly visible as both LTTR-MD and LTTR-ARMA methods have significantly lower average deadline miss times than the FCFS. In fact, the LTTR-ARMA has reduced the mean overrun time by almost 11 times compared to the first come first served scheduling.

These results show that faced by the inevitable missing of the requested deadline due to the lack of resources, predictive approaches are still able to prioritise remaining workload to reduce the amount of mean deadline overrun. The superior prediction capability of the ARMA model enabled the scheduler to prioritise the jobs more precisely and time their execution closer to the deadline. As a result, LTTR-ARMA has managed to deliver mean overrun times almost five times lower than those of LTTR-MD.

Figure 6.6 shows deadline miss times as percentages of the actual job run time. The plot confirms the superiority of the predictive scheduling methods, and in particular the LTTR-ARMA approach. The importance of these measures is in weighing the amount of scheduling bias placed on the long running jobs. The absolute value (in seconds) of the deadline overrun time could have simply been reduced by ensuring very long jobs do not miss their deadlines at the cost of penalising shorter jobs. However no such bias was detected, as shown by improvements in this scale insensitive metric.



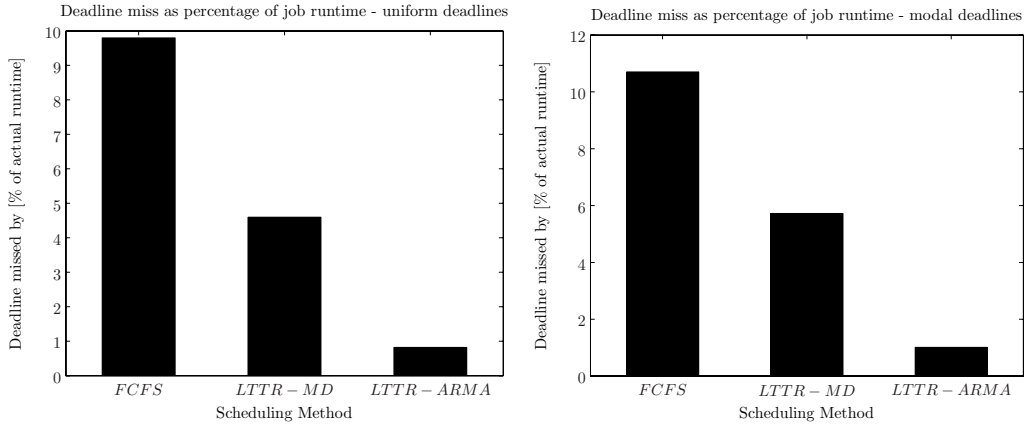
(a) Uniform deadlines (b) Modal deadlines

Figure 6.5: Comparison of central tendencies in absolute terms (seconds) of deadline overruns using mean values. Predictive methods exhibit significantly lower average overrun with both uniform and modal deadlines

Location and Dispersion of Deadline Breakage Times

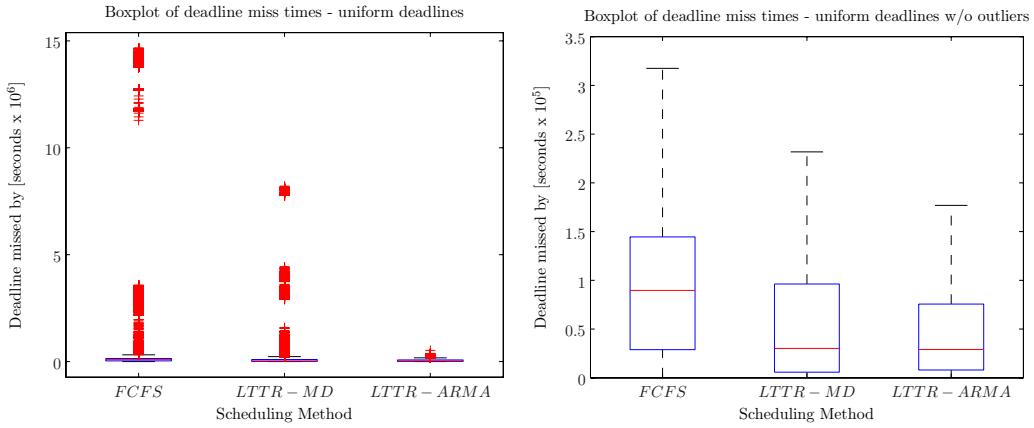
In a data set with outlier values or skew, the mean is often a poor representation of the central tendency of the distribution. From the scheduling performance perspective, the presence of these extreme values and asymmetry in the deadline overrun times is a negative characteristic reducing the reliability of deadline adherence.

Box plots of the deadline overrun times for the uniform deadline distribution, given in Figure 6.7(a), show a significant reduction in the number and scale of



(a) Uniform deadlines (b) Modal deadlines

Figure 6.6: Comparison of central tendencies in relative terms (percentage of actual job times) of deadline overruns using mean values. Predictive methods exhibit significantly lower average overrun with both uniform and modal deadlines. Comparison with the absolute terms plot reveals no bias towards short running jobs.



(a) All missed deadlines (b) Missed deadlines w/o outliers

Figure 6.7: Comparison of location and dispersion of deadline overruns in absolute terms (seconds) between scheduling methods for uniform deadline distribution. Figure (b) shows a zoomed in view of the same data without outlier values. Predictive methods exhibit significantly lower median values, less extreme outliers and smaller interquartile range than the non-predictive FCFS method.

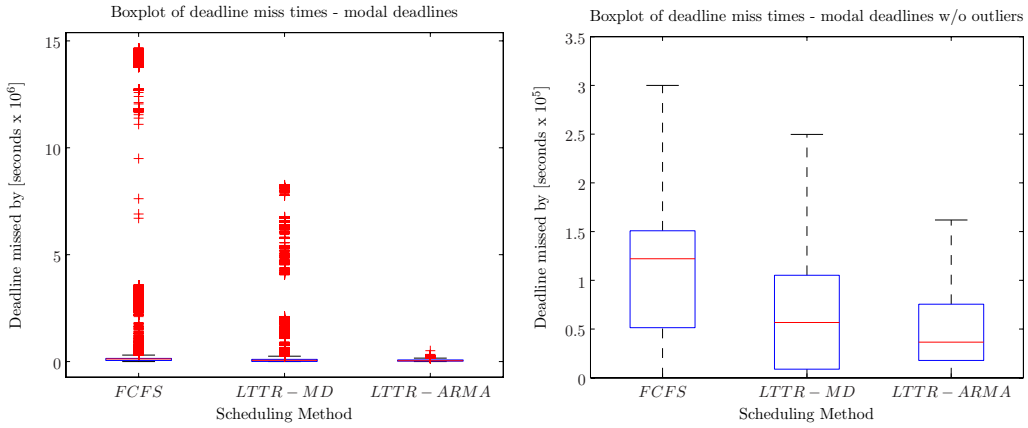
extreme outlier values between the FCFS and the LTTR-MD. The LTTR-ARMA was especially successful, with very few remaining outliers close to the upper quartile of the distribution.

To better judge the medians and interquartile ranges of the overrun times for the three scheduling methods, the box plot was redrawn in Figure 6.7(b) with the outliers removed. Again, LTTR-ARMA performs best with the lowest central tendency and the tightest value distribution. It is also the least skewed of the considered approaches, with its mean and median most closely matched.

A similar set of plots in Figure 6.8 examine the deadline miss times for the modal deadline distribution. The behaviour of the scheduling methods is very similar to the uniform model, with somewhere higher medians due to the more demanding deadline model. The combination of the predictive scheduling and good forecasting performance in the LTTR-ARMA approach leads to the lowest number of outlier values, lowest median and the lowest dispersion amongst the methods considered.

6.3.3 Distribution Functions of Deadline Adherence

Previous metrics have mainly dealt with the deadline misses and the overrun times, the most important performance aspects of the deadline scheduler. However, the amount of spare time left to the requested deadline following a job's completion is another measure of the efficiency of the scheduler. While a certain amount of such slack is desirable to avoid over-reliance on the accuracy of predicted job execution times, large amounts of spare time could be an indication of either poor runtime forecasts or poor ordering of jobs by the scheduler.



(a) All missed deadlines

(b) Missed deadlines w/o outliers

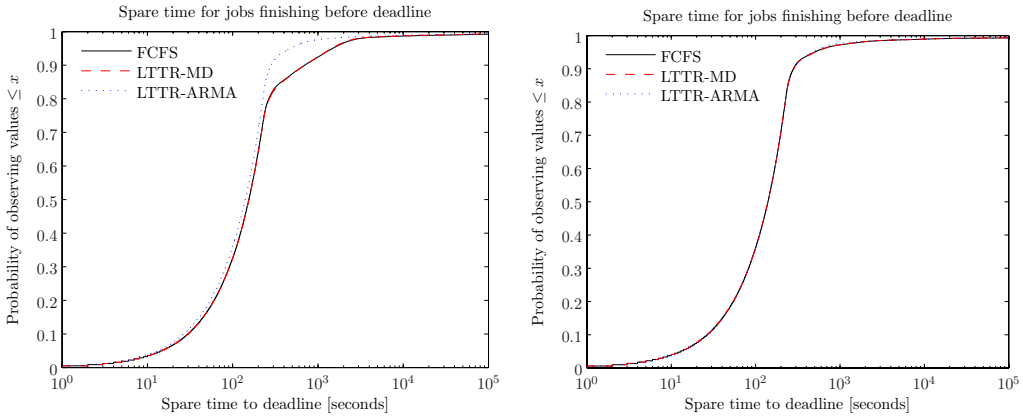
Figure 6.8: Comparison of location and dispersion of deadline overruns in absolute terms (seconds) between scheduling methods for modal deadline distribution. Figure (b) shows a zoomed in view of the same data without outlier values. Even for more challenging modal deadlines, predictive methods show vastly superior performance compared to non-predictive FCFS method.

Figure 6.9 shows the distribution function of spare time to deadline for the uniform and the modal deadline distributions. Analysing the uniform deadline plot, the distributions for the FCFS and the LTTR-MD approaches are almost identical. The LTTR-ARMA curve has a slightly steeper slope and a better tail-off characteristic indicating that 95% of jobs finish with less than 500 seconds of spare time compared to 1450 seconds for the LTTR-MD and the FCFS approaches.

In the case of modal deadlines, the spare time distributions of all three scheduling methods are almost identical. One of the reasons is certainly the more demanding deadline model, and the loss of sensitivity due to the uncorrelated and modal nature of the deadlines.

The cumulative distribution functions of the deadline overrun times are given in Figure 6.10. Both predictive scheduling methods have steeper slopes than the FCFS indicating better adherence with lower overruns for any given probability percentile. In particular, the LTTR-ARMA curve does not suffer from a long tail behaviour which was the cause of numerous outliers in the other two scheduling methods.

Similar performance benefits from the use of predictive scheduling are evident with the modal deadline distribution. These plots are also valuable in considering the effect that "softening" the deadline would have on the fraction of made deadlines. For example, a "safety factor" of 1000 seconds applied to modal deadlines would, in case of the FCFS scheduling shift another 10% of the jobs from missing the deadline to making it. But for the predictive approaches, the same safety margin would have caused over 30% more jobs to make the deadline.



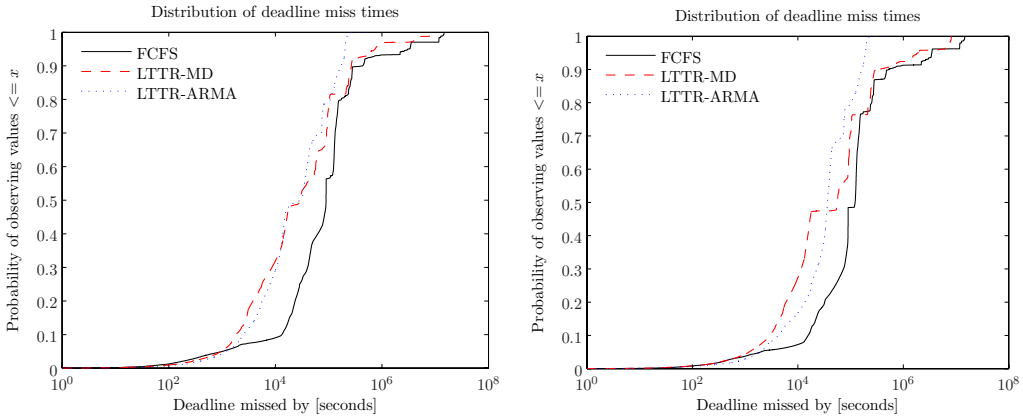
(a) Uniform deadlines (b) Modal deadlines

Figure 6.9: Comparison of cumulative distribution functions of deadline spare time in absolute terms (seconds) between scheduling methods for uniform and modal deadlines. Compared to other considered approaches LTTR-ARMA method exhibits best just-in-time scheduling performance with lowest amount of deadline spare time.

6.4 Chapter Summary

The chapter has analysed the deadline adherence performance of a novel predictive scheduling algorithm dependent on the job runtime forecasting system developed by the author. The simulation methodology has looked at the influence of the accuracy of job execution time predictions on the deadline overrun times, and the sensitivity of those values to the deadline generation model used.

The results have shown that, despite the resource starvation and subsequent



(a) Uniform deadlines (b) Modal deadlines

Figure 6.10: Comparison of cumulative distribution functions of deadline overruns in absolute terms (seconds) between scheduling methods for uniform and modal deadlines. Compared to other considered approaches LTTR-ARMA method exhibits shortest tail-off and hence least amount of outlier values.

mass missing of the deadlines, the latest time to run (LTTR) predictive scheduling method managed to greatly reduce the amount of deadline overrun compared to the common first-come-first-served batch scheduling method. The value of accurate job execution time predictions was underlined by the LTTR-ARMA method which delivered best overall performance the with lowest average overrun times (both mean and median), smallest dispersion of overrun values (very few extreme outlier values and smallest interquartile range), and the best queue optimisation with the smallest amount of slack time.

Chapter 7

Related Work

The literature survey given earlier in Chapter 3 offered an overview of the previous research work relevant to the Grid scheduling, predictions of the resource's performance and job metrics, characterisation of distributed system's workload and other related topics. The purpose of this chapter is to compare and discuss the approaches, methods and findings of this thesis to those of the most recent and most similar work by other scientists.

7.1 Workload Characterisation

The majority of the past distributed workload characterisation studies have been done based on a limited number of traces collected in the 1980-90s at the legacy parallel clusters and deposited in the Parallel Workload Archive*. While these are useful as a general starting point for research into the properties of the Grid workload, the specific design issues and resource management policies (already discussed in Section 2.1) of the Grid suggest these characterisation studies are not sufficiently representative of the likely load presented to a utility compute Grid. This section will therefore only treat the most recent attempts to characterise Grid workload based on the traces collected by other researcher in the period 2003-2005 (made publicly available in 2006) and compare them to the findings of this thesis.

Hui Li, David Groep and Lex Walters have in [26] studied a 2003, 12 month trace from the Distributed ASCI Supercomputer 2 [212] (DAS-2), a research Grid facility made up of homogeneous commodity hardware. The purpose of this study was to model the workload characteristics and enable the evaluation of different scheduling approaches. Li has found the facility to be highly underutilised with average load between 6-22% which, compared to a production facility such

*http://www.cs.huji.ac.il/labs/parallel/workload/

as the CCC with load exceeding 80%, to some extent trivialises the resource management and scheduling process.

DAS-2 job arrivals show a pronounced weekly and daily cycle with peak submissions on Wednesday and between 09:00 and 19:00 hours, while the yearly and monthly cycles are not clear. Both of these findings agree with the arrival process observed at the CCC. Analysing the job parallelism, the author's have found ambiguous correlation to the job runtime, and have observed the previously reported tendency for power-of-2 requested CPU values, although 62% of jobs require only one or two CPUs. This is a strong indication of the presence of serial jobs as the local policies on DAS-2 prevent assignment of different jobs onto different CPUs of the same dual-CPU machine.

The job memory usage was low and very modal (due to the use of shared libraries) and strongly correlated to the job runtimes. While the same was found to be true in the CCC trace, the nature of the memory usage monitoring generally prevents this information to be used for ex-ante predictions of the execution times. Jobs were found to run between 374 and 2427 seconds, very modest compared to the CCC's span of seven orders of magnitude, and attributed to the research nature of the facility. Regardless of this, the reported coefficient of variation (CV) of job runtimes is up to 16. As a consequence, the suggested Weibull and log-normal distributions does not provide a very good fit when applied to the non-partitioned workload.

The study briefly looks at the user behaviour finding repetitive behaviour in the submission of jobs: a small number of applications are run very frequently and a much larger fraction jobs are run just once. Contrary to the numerous other reports previously discussed, Li finds significant correlation between the actual job runtime and the user's requested time. Overall, the choice of the characterisation approach and studied metrics taken by Li supports and validates those taken by the author of this thesis. The analysed trace however seems to lack the diversity and dynamics of a production environment such as the CCC.

Alexandru Iosup, Hui Li, Lex Walters et al in [157] build on their previous work by examining traces extending over six or more months of the year 2005 from three production Grids (LCG [213], Grid3 [214] and TeraGrid [215]) and an academic research Grid (DAS-2 [212]). The work aims to offer a general insight into how today's Grids are used and help in designing the infrastructure and services for future installations. Additional focus of the work is in quantifying the fairness of the delivered scheduling and the level of user satisfaction. The work concludes with a discussion of the data collection problems on the Grid and calls for a better integration of the Grid resource monitoring systems, much in line with the author's arguments for developing an extension to the Ganglia monitoring system (see Appendix A.2).

The authors report high utilisation levels (60-80%) on production systems,

and a low load of only up to 10% on the DAS-2 research Grid. Such findings confirm that the CCC, with its utilisation of over 80% is indeed a very highly loaded, and thus difficult to schedule system. The observed arrival process is strongly influenced by the weekly and daily cycles, while the inter-arrival times are very bursty and indicative of "bag-of-tasks" submissions. Memory use on these systems is also reported to be highly modal.

Analysing the job execution times, the study has found production facilities running much longer jobs (with the mean of $\approx 15,000s$, and the 95^{th} percentile of $\approx 60,000s$) than the academic ones (with the mean of $\approx 350s$, and the 95^{th} percentile of $\approx 600s$). The CCC job runtimes are therefore similar to those reported for the other production Grids, as was shown in Figure 4.15 on page 83. Iosup also reports that an overwhelming fraction of jobs on the production Grids are of either serial or "embarrassingly parallel" type requesting a single CPU and requiring no synchronisation with the other job instances. Even on the DAS-2 research grid, they report the number of serial jobs submitted increasing tenfold in two years.

Considering user behaviour, the authors of this characterisation study have noted the so called 10/90 phenomenon with a small number of users submitting largest numbers of jobs and a small number of jobs responsible for largest fraction of the CPU usage. The workload was also evolving over time, and this was evident at the system, VO and user levels. Both of these findings are consistent with the behaviour observed at the CCC which was crucial in developing the approach presented in this thesis.

Emmanuel Medernach in [27] examines a 10 month, 2005 LPC cluster in the EGEE Grid [29] workload in the context of modelling (using Markov chains) and simulating different scheduling policies. The cluster considered is a homogeneous, space shared installation and a part of the EGEE infrastructure. The workload is analysed with respect to two partitioning metrics, VO owning the job and the queue to which the job was submitted. The workload consists of only two user applications and regular administrative jobs and therefore compares poorly to the diversity found in the CCC trace.

Medernach analyses the arrival process and observes a daily cycle with a spike of job submissions at full hours due to the repetitive and automated submission of the administrative test jobs. Arrivals are non-Poisson, bursty and with a high CV value. In examining the job queue times, Medernach observes their very high CV value (≈ 22) and comments on the wide variation of waiting times experienced by different VOs, and the blocking of shorter jobs by the very long running ones. This leads to the suggestion that a measure of "relative urgency" would be beneficial, and further motivates the deadline approach taken in this thesis.

Considering the job execution times, this study finds that a general model

spanning the entire distribution is unlikely and proposes a high order (3-6) log-uniform one. The author has found user predictions of the job runtimes to be inaccurate and generally uncorrelated to the actual execution times. Importantly, Medernach has found job execution times are strongly autocorrelated, thus confirming the CCC findings and supporting the time-series forecasting approach taken in this thesis.

Menno Dobber, Rob van der Mei and Ger Koole in [28] examine the execution times of compute-bound jobs on the PlanetLab [216] space and time shared heterogeneous academic research Grid. Unfortunately, the workload is synthetic, generated by the authors running consecutive and identical tasks and is therefore of very limited use in studying the usage of production Grid clusters.

Regardless, they have found that, due to the process preemption on the time shared hardware, the job runtime distribution is bursty and with many high value outliers - suggesting a long-tailed effect may be present. Dobber observed great variability of the runtimes indicated by a high CV value, and their strong autocorrelation leading to a more pronounced long-term fluctuation.

Summary

The characterisation of the Grid workload is still very scarce due to the novelty of the Grid technology and the limited amount of the available production Grid traces. Presented work supports the views taken in this thesis that a utility compute Grid would be a space shared, non pre-emptive, homogeneous resource on the individual cluster level. The majority of the characterisation work investigates the properties of the job arrival cycle and the job runtimes, reporting on their value distributions, seasonality, and variability (by using the coefficient of variation metric).

	Job arrival cycle			Modal Memory Use	Utilisation
	Daily	Weekly	Bursty		
Li	●	●	●	●	5-10%
Iosup	●	●	●	●	60-85%
Mendernach	●		●		
Lazarević	●	●	●		89%

Table 7.1: Comparison of related Grid workload characterisation research with respect to job arrival patterns, job memory allocation and overall system utilisation.

A summary of the findings by the reviewed work relating to the job arrival cycle and the overall facility utilisation is given in Table 7.1. The daily cycle was reported in all of the workloads, and the weekly in all but one. The utilisation

varied significantly depending on the nature of the facility but was generally under 10% for academic installations and over 60% for production Grids.

The overview of the statistical properties of the job runtimes, job queueing times and the fraction of parallel jobs for the reviewed characterisation studies is given in Table 7.2. Clearly, the range of job execution times varies significantly between the Grid installations, and is an important aspect into their target use. A very short maximum job runtime, like those reported by Li and Dobber, imply that those Grids are mostly used as testbeds and are not representative of a more complex workload expected at a production facility.

The tendency of the users to almost exclusively submit sequential jobs is supported by all of the listed studies, which also unanimously report the log-normality of runtimes and their high variability. The properties of the queue wait times, where reported, are also characterised by a high variability. Their other statistical properties, including their central tendency, are highly conditional on the arrival process and the distribution of job runtime values.

	Job execution time				
	Range (s)	CV	Distrib.	Parl. jobs	Queue time
Li	< 2500	$5 - 16$	log-normal gamma weibul	38%	
Iosup	$< 5 \cdot 10^5$	$2 - 12$	log-normal	$\approx 0\%$	modal short
Medernach	$< 1.7 \cdot 10^5$	$3 - 12$	log-normal	0%	high CV varies between VOs
Dobber	< 120	$0.2 - 1.8$	long-tail multimodal autocorrelated	0%	
Lazarević	$< 10^6$	$0.6 - 15$	log-normal long-tailed autocorrelated	0%	high CV long-tailed

Table 7.2: Comparison of related Grid workload characterisation research with respect to job execution time, degree of job parallelism and queue wait times.

Overall, the survey of the closely related Grid workload characterisation research supports the findings of this thesis and highlights its distinct contributions in the analysis of the evolution of job properties and their temporal characteristics. The need for further studies on the usage statistics of the real-world, production Grids is clear and motivated by the importance of the load characteristic in all stages of Grid system planning, provisioning and management.

7.2 Job Execution Time Forecasting

From the survey of previous work given in Chapter 3, it is evident that various predictive techniques were extensively used to forecast the dynamic properties of the distributed computing systems, such as the network performance, host load or available memory. Different approaches were also suggested for prediction of the execution times of distributed computing jobs and the closely dependent metric of queue wait times and job start times. In this section, the focus will be on comparing the work presented in this thesis to the most recent and relevant research that uses historical Grid utilisation to predict future job execution times.

In this context, it became clear from workload characterisation experiences that for all but the trivial workloads, jobs must in some way be grouped or partitioned into similarly behaving clusters before attempting to fit them with a predictive model. The primary comparison between this thesis and the previous work will therefore be based on the two following aspects: the metrics and the methods by which the entire workload is partitioned, and the actual forecasting algorithms used to make the predictions.

Warren Smith, Ian Foster and Valerie Taylor in [126] focus on developing a search algorithm for job properties yielding the best similarity and predictability. The authors implement an automated discovery of partitioning metrics based on the greedy search and genetic algorithms. In line with the CCC results, they have found that the job owner and the job name are the most significant partitioning metrics. However, Smith does not look into the temporal job properties (such as the time and date of submission) but defers this for further work.

Forecasts of the job runtimes are made either as absolute values or relative to the user supplied execution time estimate. Contrary to commonly reported results, Smith has found that the use of user estimates improves the accuracy of predictions by 23-43%. These predictions were made using two prediction algorithms: MEAN - averaging over the entire history of similar jobs, and LR - linear regression over the previous job runtimes and the requested number of CPUs. Smith reports the accuracy in absolute terms (minutes) and as a percentage of mean job runtime ranging between 40-58% for genetic algorithm and 40-65% for greedy search.

Unfortunately, Smith's work is based on 12 month long traces from four parallel clusters dating back to 1995-96 which are not representative of the current Grid usage (see Section 2.1). The choice of the accuracy measures together with the unknown statistical properties of the job runtimes makes direct comparison of results difficult. The forecasting methods are simple and largely nonparametric but they do have a significant prediction error. However, the better performing genetic algorithm is generally considered computationally expensive [217] and may not be suitable for online use with extensive Grid usage histories.

Byoung-Dai Lee and Jennifer Schopf in [129] aim to predict the application runtimes on space and time shared homogeneous resources with a varying background load. They propose the use of "filters" to generate subsets of similar job runs and resource conditions based on the application input parameters, degree of job parallelism and "resource capacity" metrics such as the machine load, network bandwidth and latency. Predictions are generated using a linear regression algorithm and its accuracy reported using normalised percentage error.

Presented results show a significant improvement in forecasting performance when a filter is applied, reducing the average error from almost 50% to between 20% and 30%. However, the selection or the number of filter criteria does not reduce the error any further. From a number of offered resource status metrics, the measure of background load is the dominant one consistently leading to the best predictions.

Despite returning more accurate runtime predictions using a similarly simple forecasting algorithm as Smith, Lee significantly limits his scope, and thus applicability of his approach, to applications with deterministic runtimes influenced only by their input parameters and not by the distribution of the input data set. The performance of Lee's prediction method is evaluated by using only two custom applications run separately, which is hardly representative of the real-world workload reported on the production Grids.

Hui Li, David Groep, Jeff Templon and Lex Wolters in [218] predict job execution times in the context of queue wait time forecasts. The work is based on a 3 month 2003 trace from the NIKHEF cluster of the European Data Grid facility [219]. They consider partitioning the workload using all metrics available in the standard accounting records such as the job's submitting username and VO, the name of the job and its submission queue as well as the number of requested CPUs. The grouping metrics are selected using an undisclosed and undocumented heuristic approach which has excluded the degree of parallelism and the job name as parameters providing no extra categorisation information. No temporal metrics have been considered or used in partitioning the workload.

Li implements a windowed mean (WM) and linear regression (LR) forecasting algorithms and undertakes a limited quantitative analysis to choose the best order for these. He concludes shorter windows sizes are better and selects WM(1) and LR(5) as the predictors. The accuracy measures are simply reported in absolute terms, as the average error in seconds, and as the percentage value of the average job runtime. These errors were in the 14-35% range. Li has also implemented a simple "expert system" which selects the next forecast based on the error values made in the previous prediction step.

This work is a noteworthy attempt at generating job runtime predictions based on the historical information, and had produced usable results. The NIKHEF workload is not widely used and analysed and, by the very limited

information provided by Li (average runtime of 4672 and 11537 job entries), it is not possible to conclude how deterministic the workload is and whether its statistical properties are indeed representative of a production utility Grid. The heuristic used for the job partitioning has not been discussed, and it is unclear if this process is automated and adaptable to different usage patterns. The exclusion of the job name metric is contrary to the CCC findings where such information, although not always available, was shown to be of good use. Li has used simple prediction algorithms, and while they gave reasonable accuracy, the process of their parametrisation seems opaque. As with other related work, the offered accuracy measures are not directly comparable and should only be considered together with the workload used. In his most recent work [220], Li has looked at using the genetic algorithms at the workload partitioning stage and implementing instance based learning [221] runtime predictors.

David Talby, Dan Tsafrir, Zviki Goldberg and Dror Feitelson in [222] aim to replace the user runtime estimates in backfilling* FCFS schedulers with system-generated predictions. The work is an extensions of Tsafrir's simple forecasting method (presented in [158]) of averaging the runtime of the last two jobs submitted by the same user. For grouping of similar jobs, Talby uses the degree of job parallelism, the user's runtime estimate and the executable name. But the proposed matching algorithm requires an explicit and ordered list of these criteria to be supplied. The work also proposes a novel partitioning algorithm based on the concept of "user sessions": continuous temporal periods of per-user activity which were formalised by Zilber in [223] and found to have reduced variance between submitted jobs. Talby attributes jobs to the same session if the think time† between them is less than 20 minutes, a value taken from [223].

The prediction algorithm is a simple median of the last three jobs matching the similarity requirements. The accuracy measures used are relative to the author's previous implementations and are very difficult to interpret and compare. Contrary to their starting point in Tsafrir's work [158], the authors strongly favour job similarity over recency.

Although this work is based on an extensive workload collection of over 400,000 jobs from four different parallel computer sites, these are likely to have significantly different statistical properties than modern Grid installations. The interesting approach of user sessions offers strong support for considering the temporal characteristics and the evolution of the workload as was done in this thesis. Despite this, Talby's work is dependent on too many arbitrary parameters to be truly applicable in the context of an automated, utility computing environment.

*The optimisation process queueing smaller and shorter jobs ahead of the larger ones which are unable to start due to insufficient resources.

†Defined as time between the termination of the previous and the submission of the next job

Peter A. Dinda has in [128] introduced a Running Time Advisor system for predicting the execution times of compute bound, moldable and interactive virtualisation applications on homogeneous space and time shared distributed systems. The basis of this work is Dinda's previous seminal research into the prediction of host load using time-series models [107].

Due to a very specific and narrow scope of the applications whose execution times it is intended to predict, Running Time Advisor does not attempt to group the jobs into similarly behaving groups. Instead, the predictions of the running time of a task are computed from the prediction of the host load and the nominal execution time of a task on an unloaded host. Therefore, the Running Time Advisor effectively predicts the slowdown an application of a known execution time will experience due to the background load on the worker node.

Dinda's work on the prediction of the host load and his use of time-series analysis and forecasting methods were a significant inspiration for the work presented in this thesis. However useful the presented approach could be within a specific and limited domain, the algorithm's dependence on the nominal job execution time (which is either supplied explicitly or measured by running a job on an unloaded worker node) makes it incompatible with the utility Grid environment.

Richard Gibbons and his Historical Application Profiler [152] is often quoted as the first work in the context of the job runtimes predictions based on the historical information. Gibbons has established the basis for the use of job properties, and the coefficient of variation of their runtimes, to partition the workload into more predictable sets. For that purpose he used the job name, owner's username, the degree of parallelism and the time the job has already executed for at the time of making the prediction (job age). These metrics were manually combined into six static templates used for making forecasts.

Gibbons used the mean of previous job runtimes as the primary forecasting algorithm, applying a liner regression over the number of requested CPUs if such number of nodes has not been requested before. The Historical Application Profiler was tested with a synthetic load consisting of 200 jobs submitted with an exponential inter-arrival times with the mean of 150 seconds. Later Grid workload characterisation studies, including the one given in this thesis, have found this not to be a representative behaviour. The same data indicated that the mean is not a reasonable predictor due to the extensive skew present in the distribution of job runtimes.

Allen Downey in [109] focuses on the prediction of queue wait times based on the forecasts of the remaining job execution times for the jobs queued. The approach was tested on the 1994-96 traces from the SDSC Paragon [224] and CTC IBM SP2 [225] space shared homogeneous parallel clusters. Workload partitioning was done only on the basis of the scheduler queue to which the job was submitted.

Downey proposed a technique that categorised all applications in the workload and modelled the cumulative distribution functions of their execution times. The predictions were then made either using the median lifetime model (given a certain age of the job) or a conditional average lifetime. These techniques perform best in predicting how long a job will run considering it has already executed for a given amount of time. Downey primarily used those forecasts to predict the time until n additional CPUs will become available leading to unblocking of queued parallel jobs.

Downey's work was the first to report on the log-normal distribution of the job runtimes, a property, also found in the CCC workload, which he continued to examine in [25]. His prediction methods, although simple and effective, were found not to be well suited to estimating the runtime of jobs at age zero, in other words while they are pending in the queue [126]. An often raised critique of this work is that Downey has used the entire trace to parametrise the distributions subsequently used to make the forecasts of the very same workload. The reliance on the user's selection of the submission queue as a single metric for defining the job similarity leads to a significant degradation of runtime prediction accuracy as the user's estimate of the job execution time (and thus his selection of the submission queue) worsens.

Summary

The survey of the most closely related work treating the job execution time predictions based on the historical information showed all of the approaches attempted to group the jobs into partitions or clusters of similar behaviour in order to reduce the variance of the job runtimes and facilitate the prediction using their selected statistical forecasting method. For this purpose, the majority of the work uses a few basic job properties and, with the exception of the Talby's session based approach, none makes use of the temporal information associated with the job. Methods of workload partitioning range from trivial fixed sets to the computationally expensive genetic algorithms but are all too often not based on a rigorous examination of the relationships between the job metrics found in the representative Grid traces. An overview of the partitioning metrics used by the author and fellow researchers is given in Table 7.3.

The most popular forecasting algorithms are based on the estimation of the central tendency of a group of similar jobs using either mean or median predictors. Linear regression is another often used technique, and was combined with the job degree of parallelism property to exploit its relationship to the runtime found in some of the workloads. Despite the overwhelming evidence that the job execution times are auto-correlated no previous work has suggested or attempted modeling them using any time-series methods similar to those presented in this thesis. An overview of prediction methods used is given in Table 7.4.

	Workload partitioning methods							
	VO	User	Job name	Parall.	Queue	Arg.	User Est.	Temp.
Smith		●	●	●	●	●		
Lee				●		●		
Li	●	●	●	●	●			
Tsafrir		●						
Talby			●	●			●	
Dinda								
Gibbons		●	●	●				
Downey					●			
Lazarević	●	●	●					●

Table 7.3: Comparison of related job execution time forecasting research with respect to workload partitioning methods used to define "similar" jobs. Shown job properties are submitting VO and username, executable or job name, degree of job parallelism, queue name to which the job was submitted, command line arguments passed to the job, user's estimate of job runtime and temporal properties such as time of submission.

	Prediction algorithms							
	Mean	Median	Min-Max	LR	ES	AR	MA	AR(I,F)MA
Smith	●			●				
Lee				●				
Li	●(W)			●				
Tsafrir	●(W)	●(W)	●(W)					
Talby		●(W)						
Dinda						●	●	●
Gibbons	●			●				
Downey	●	●						
Lazarević		●			●	●	●	●

Table 7.4: Comparison of related job execution time forecasting research with respect to statistical prediction methods used. Shown predictors are mean (windowed), median (windowed), minimum - maximum (windowed), linear regression, exponential smoothing, auto-regressive, moving average and a family of auto-regressive integrated fractional moving average methods.

7.3 Deadline Scheduling on the Grid

Research activities in the Grid scheduling field closely reflect the popularity of the backfilling FCFS schedulers and mostly deal with the incremental improvements of such algorithms. Although the concept of deadline scheduling is a well researched topic in the real-time systems, it has seldom been considered in the context of scheduling jobs on the distributed platforms such as the Grid. This section introduces previous work that has attempted to deliver scheduling to a user requested deadline, and discussed their relevance to the methods given in this thesis.

Atsuko Takefusa, Satoshi Matsuoka, Henri Casanova, Francine Berman in [226] focus on scheduling of independent serial jobs in the multi-client multi-server environments such as the network-enabled servers (NES [18]) and the computational Grids. The aim of the work is to minimise the overall occurrences of the deadline misses and their magnitude while enabling the users to make a tradeoff between the deadline adherence and the computational cost.

The proposed algorithm computes the job processing time by dividing the logical computational cost (in some arbitrary units) with the resource service rate, multiplies the time to deadline by a "tuning" factor quantifying the conservatism of the scheduler and looks for a suitable worker node that can either satisfy the deadline or, if none are found, minimise the amount of the deadline overrun. By using their Bricks tool [227] to simulate the deadline scheduling algorithm and the submission of jobs onto a virtual heterogeneous, space and time shared set of resources, Takefusa has confirmed that his algorithm delivers better deadline adherence than the reference greedy approach.

However, this simulation has used a synthetic workload and made some important simplifications to the properties of both jobs and resources. The client to server ratio was one to one, the network and server performance levels were randomly drawn from a uniform distribution with a modest range, and the background load was fixed at 10% of the node's capacity. More importantly, the job duration was drawn from a uniform distribution with the execution times of 5 to 60 minutes and a Poisson arrival process with an average inter-arrival time of 60, 90 or 120 minutes. Such distribution of the execution times and the level of utilisation does not create a scheduling environment as challenging as those found in the current production Grids. The deadlines are generated by multiplying the actual job runtime by a factor drawn from a uniform distribution between 1 and 3. Such deadline generation methods was found not to be representative of the way users are likely to specify deadlines [207] and shown in this thesis to be less demanding of the deadline scheduler.

Regardless, the work by Takefusa introduces the concept of the job deadlines to distributed computing, linking it to the notion of the Grid economy and

supporting its use as a measure of the job urgency. The proposed scheduling algorithm, despite its reliance on execution time forecasting methods which can hardly be implemented in a general-purpose production Grid, demonstrates the possible benefits of the predictive deadline scheduling approach. The work was the basis for some incremental improvements to the runtime predictions and fallback methods done by Caron in [228].

David Abramson, John Giddy and Lew Kotler in [229] build on the Nimrod [99] and Nimrod/G [98] tools to deliver soft-deadlines to the parametric study applications. These jobs, consisting of independent tasks, can be considered moldable as they can be run on an arbitrary number of processing units. The goal of the scheduling process is then to dynamically select the size and membership (in terms of the computational performance and the price) of the resource pool to ensure the overall job completion prior to the requested deadline and at the requested monetary cost.

The authors have demonstrated a good deadline adherence performance of the Nimrod/G scheduler, and its commercial version Clustor, using a number of specialised applications in the field of bio-informatics, ecological modelling and computer aided design areas. However crucial to such success is the highly deterministic and predictable execution time of each independent task, and the ability to dynamically change the job's degree of parallelism in order to speed the execution up or slow it down. While the parameter sweep application targeted in Abramson's work form an important part of the scientific workload, they are not representative of a general-purpose compute load likely to be presented to a utility Grid.

In this work, Abramson strongly embraces the economic aspect of the deadline driven scheduling as the necessary lever to control the selection and utilisation of resources. The deadline is also strongly favoured as a way of expressing the user's view of the job urgency and priority.

Peter Dinda has in [131] extended his previous work on the host load prediction and job runtime estimation by implementing an advisory system that recommends the execution host based on the job's soft deadline and the CPU requirements. The work is limited to the same scope of interactive, compute-bound, moldable visualisation applications and requires that the nominal execution time of each application on an unloaded system is known in advance.

The presented Real-time Scheduling Advisor is tested using a synthetic workload consisting of jobs arriving consecutively with a uniform think time distribution between 5 and 15 seconds and a nominal execution time uniformly distributed between 0.1 and 100 seconds. Clearly, with such arrival and runtime statistics, and with no queued jobs, the workload is not representative of a Grid installation like the UCL's CCC.

Dinda's extensive work on the resource performance predictions, their integration into the forecasts of job runtimes, and the deadline schedulers were strong motivating factors for many subsequent researchers, but is of very limited applicability to the defined scope and target platform of this thesis.

Summary

The limited amount of previous work on the topic of deadline scheduling for the distributed computing systems that was presented in this section establishes the feasibility of the approach and confirms its added value. The concept of the job deadlines is closely related to that of a computational economy: all surveyed work makes provisions for such systems and the inherent tradeoff between the cost incurred by the user and the guarantee of the deadline adherence. Finally, job deadline is confirmed as the most appropriate measure of the urgency of each job submitted by the user.

However, there are numerous opportunities for further work in this area. The availability of job execution time forecasts, as delivered by the work presented in this thesis, makes numerous advanced scheduling methods used in (near) real-time systems portable to the Grid environment. Coupled with a barter or a bidding economy model, the possibility for a truly global computational market exists, on which compute resources will be traded and used like many of today's commodities.

Chapter 8

Open Questions

In presenting the findings and results so far, the thesis has focused on justifying its approach and presenting its methods and the obtained results. This chapter will take a critical view and discuss the most challenging aspects of the design, development and testing stages of the deadline scheduling system presented. Ideas for improving these will be given as the basis of further work that the author, or other researchers, may engage in.

8.1 Workload Characterisation

Considering the necessity to analyse a large amount of Grid usage data, and the reliance of this process on statistical tools, the workload characterisation aspect will continue to benefit from the developments in the fields of exploratory data analysis, data mining techniques and clustering algorithms.

Representativeness of the dataset, in this case of the workload trace, is an often raised issue in approaches that develop models based on statistical analysis. Unfortunately, due to privacy laws, intellectual property legislation and many other reasons, good quality workload traces of sufficient duration are hard to come by. This is even more so in the case of a novel technology such as the Grid as it takes several years for the production grade facilities to go online and for a reasonable amount of data to be collected. Scope for further work will be in using these new workload traces to perform additional characterisation studies and comparatively analyse them with legacy high performance parallel workloads.

In its twelve month duration, the CCC dataset analysed in this thesis contained 37 users belonging to 27 Virtual Organisations executing a diverse set of over 2000 different job names. These properties, and a comparison with previously studied workloads (given in Section 7.1), strongly suggest it can be considered as a representative example of a workload likely to be presented to a utility

Grid cluster serving a diverse population of users. To further support this, an additional study of the only other publicly available Grid workload at the time of writing was undertaken and presented in Appendix B.

Due to the way observed workload features are used later in the job execution time predictions, the entire approach is much less sensitive to the trace representativeness than it may initially appear. In previous workload characterisation research, the focus was on creating generative models by capturing the behaviour of a certain workload metric with as few parameters as possible. The model is therefore under risk of locking onto specific features of the workload not representative of a broader behaviour. The characterisation study in this thesis does not need to pre-define any models as each of the forecasting methods used trains on the actual historical usage data of the cluster whose workloads it is to predict. This reduces the threshold of the required representativeness to the support of the assumption that job temporal- and meta-properties have a sustained and correlated relation to the job execution times.

"Random" or uncharacteristic work that is not autocorrelated, or that could not be modelled with a reasonable accuracy, was present to a varying degree in the job partitions based on one, two and three clustering job properties. While this is expectable and certainly leads to a reduction in the overall forecasting accuracy, more problematic was the presence of few partitions containing only jobs with a seemingly random execution times. Unless these could somehow be further partitioned using as yet an unavailable metric into a more manageable set, the predictive scheduling approach would not yield acceptable results for those jobs.

This issue, although noticed on a very limited scale, does offer the scheduler the ability to differentiate the "badly" behaved jobs before they are run by the combination of their job properties. Therefore, jobs submitted by a certain user running a certain application can be segregated from the rest of the workload and handled differently, either by running them on a dedicated pool of "best effort" machines or by applying a different set of Grid economy policies.

Availability of monitored metrics and the overall transparency and compatibility of the accounting and usage data records poses a big challenge for the entire Grid community. Despite the efforts within the Open Grid Forum and the Usage Records Working Group *, the author has faced many problems in acquiring the necessary usage statistics. Clearly, a more detailed and a more granular historical data holds a higher potential of discovering functional dependency between the job properties and its resource usage, and could thus lead to much improved forecasts. As this thesis has shown, even the three or four basic pieces of job information, when used appropriately, could lead to satisfactory performance.

*http://forge.gridforum.org/sf/projects/ur-wg

Presently, usage records lack the ability to uniquely identify the application being run and the parameters passed to it, information which could lead to a significant improvement in the accuracy of the job execution time predictions. In the future, a Grid workflow manager could uniquely hash executable files, their parameters and input data sets bringing more transparency to the presently used generic deployment scripts, and enabling the predictive schedulers to identify changes in the applications being run.

8.2 Job Execution Time Forecasting

Previously, predictions of job execution times have either been provided by the submitting user, or derived through application instrumentalisation. The willingness and the ability of the users to supply reasonable forecasts seems to have been overestimated and is not likely to be pursued any longer [116]. The only currently foreseeable competition to the historical modelling approach is likely to come from some form of application instrumentalisation. This technique was previously used for high-value applications or specialised hardware, but has not been widely adopted due to the extensive human work needed to instrument and recompile software on different execution platforms. However, automatic instrumentation tools (proposed in [230, 231] for example), and the increased adoption of binary compatible code and native virtualisation, helped by the support from the hardware and operating system vendors, may one day enable an efficient and portable way for an application to communicate its progress to the Grid middleware.

Time series forecasting algorithms and parametrisation techniques used in this thesis present only a selection of methods that are currently available to statisticians. More complex approaches, such as that of Autoregressive Fractional Integrated Moving Average [232] (ARFIMA) which is the generalisation of all three classes of linear time series models, may prove to be more accurate and adaptable. Analysis of new production workloads may require and justify the use of non-linear, heteroscedastic* time series models such as Autoregressive Conditional Heteroskedasticity[232] (ARCH).

The context of the work requires all these models to be in some way automatically parametrised which proved to be a challenging task. During testing, the parametrisation heuristic had to be made robust to various extreme values and exceptions in order to produce stable models. Further work on improving the way in which models are parametrised, by perhaps borrowing on some approaches used in modelling the financial time series, would certainly lead to a reduced training set requirements and an increased model accuracy.

*A sequence of random variables with different variances

The scope for further work also exists in creation of an "expert system", a technique often used in the time-series forecasts. Several prediction algorithms, differently parametrised and suited to different types of time-series, are run in parallel. The expert system tracks their historical performance in predicting each of the time series and decides which of them to base a spot prediction on.

Complexity vs. Performance trade off raises the question whether it is sensible to develop a complex model for jobs executing for only a very short period of time. As the entire forecasting engine should run in near real-time, spending time on analysing and modeling short or low-value jobs may be worse than just running them in a first-come-first-served fashion. With this in mind, the analysis of the forecasting methods in Chapter 5 offered algorithms of varying complexity, and discussion of their results revealed the trade-off in the accuracy of predictions.

Considering that the forecasting engine was a proof-of-concept implementation, the comparison of computational complexity of the presented prediction methods was deferred until production grade code is available. This further work may be undertaken as part of the commercialisation efforts described in Appendix C. However, based on the observations made through substantial testing and simulation, the performance of the forecasting models should not present a significant difficulty for any modern hardware at the point of job arrival rates well above those observed today. Comparatively, these time-series models are much less computationally expensive than some other proposed techniques such as the genetic algorithms [80] used by Song [84], Aggarwal [82], Kim [83] or Cao [233], neural networks [234], game theory [78] used by Young [79] and Beaumont [64] and simulated annealing[81] also used by Young in [79].

Initial lack of historical data on a newly deployed system, or for newly introduced users and applications, can be overcome by scheduling such jobs in a FIFO batch mode until prediction models can be fitted. The availability of different forecasting algorithms can also be exploited by initially fitting a simpler model requiring fewer training data points. As was shown in Chapter 5, even the three point moving average predictor yields usable results.

8.3 Deadline Scheduling Algorithm

By making the expected job execution time available to the Grid scheduler, the framework presented in this thesis creates an opportunity for migrating numerous scheduling algorithms and techniques from (soft and hard) real-time systems into the domain of utility computing. Combined with the job check-pointing and migration that some Grid middleware supports, a truly adaptable and dynamic platform that responds to the changing load and user priorities could be created.

Deadline feasibility was not considered as deadlines had to be generated artificially and have all been at least equal to the actual execution time of the job. This would hardly be the case in an actual production system where users would have to be in some way guided as to the costs associated with a requested deadline as well as a probability of it being met.

Judging the feasibility of a requested deadline could be done based on the forecasted execution time of the submitted job. If the deadline is within some margin of the forecast it could be deemed feasible. Being a probabilistic measure, this should not preclude the admission of the job as the forecasted execution time could be grossly over-estimated and a much shorter deadline could indeed be possible. But it could be used in combination with the Grid economy pricing policy, and perhaps a different SLA, to reduce the penalty the Grid operator would face if an over ambitious deadline is not met.

8.4 Chapter Summary

The successful implementation of the autonomous job execution time forecasting system described in this thesis has opened up the field for significant further research into scheduling systems which can make best use of this added information. As such, it has given rise to some challenging new problems and these, together with the issues faced by the author in the implementation stages of this research, have been discussed in this chapter.

Chapter 9

Conclusions

Prompted by the need for a job scheduling method that is more flexible and better suited to the human workflow, the thesis has set off to develop the necessary technologies needed to support an autonomous and self-managing scheduling system based on user supplied job deadline requirements. To this end, the thesis contributions were threefold.

To form a rigorous and factual basis on which the relevance of job properties to runtime predictions can be judged, and to gain insight in the ways a real-world general purpose production Grid is being used, the thesis has presented a characterisation study of a 12 month workload from the UCL's CCC Grid facility. As a first Grid trace of such length and such diversity, this workload confirmed the presence of the usual cyclic patterns occurring in human generated activities. Compared to previous studies of parallel and distributed workloads, this characterisation study paid special attention to the evolution of user behaviour and workload properties over different timescales and the correlation between temporal and other job properties, and the job execution times. This has shown that a significant degree of correlation exists and can be exploited for generating more accurate predictions. It has also shown that the user behaviour and workload are constantly evolving and that a dynamic and adaptable system is required to ensure adequate system modeling.

Finding that job runtimes are highly autocorrelated, self-similar and long-range dependent, the thesis has suggested applying time-series forecasting models on partitions containing similar historical jobs. An exhaustive search approach has been proposed to define pivotal job properties which, when used to partition the workload, lead to its reduced variability and increased predictability. Based on the comparison of runtime variance by using the coefficient of variation metric, the method is able to autonomously discover functional dependence between

different job properties and execution times.

By using the actual trace from a production Grid cluster, exponential smoothing, auto-regressive, moving average and auto-regressive moving average forecasting methods were compared to the benchmark windowed median predictor. The accuracy metrics were based on the best statistical practices for comparison of series with different location and in the presence of outliers. Reported results demonstrate the superior performance of the ARMA prediction method coupled to the three-dimensional partitioning of similar jobs based on the owner VO, job name and a temporal metric defining the week in which the job has been submitted.

With the ability to predict the execution time of a queued job, the thesis has introduced a deadline scheduling algorithm previously not applied in the context of distributed computing. A trace reply simulation using the actual CCC workload was used to simulate a scheduling scenario in which jobs arrive with user supplied deadlines. The simulation explored the effect of differently generated job deadlines, and the deadline adherence and overrun of the proposed Latest Time To Run (LTTR) scheduling compared to the commonly used FIFO batch scheduler. It concluded that job runtimes forecasts, of the quality delivered by the time-series based predictions and applied to the LTTR scheduling can improve deadline adherence and significantly reduce deadline overrun on highly loaded systems.

Overall, the thesis has shown that a deadline scheduling system for a utility compute Grid clusters can indeed be based on an autonomous and self-managing historical statistical prediction component that does not require any user input or any modification of user submitted application or instrumentalisation of the Grid middleware.

Appendices

Appendix A

SO-GRM Project Related Work

The following appendix presents Grid related work undertaken as part of the Self-Organising Grid Resource Management project supported by EPSRC (GR/S21939) and BT Research. Throughout the three year duration of this project the author was in charge of deploying and maintaining a Grid testbed compromising of locally networked clusters in UCL and BT's labs at Adastral Park, interconnected through a WAN link. The practical experience gained through these activities, and the involvement in the implementation of the Grid management components described in this chapter, shaped the further direction of the thesis research and reiterated the necessity for autonomous and self-organising management architecture.

The author's two primary contributions to this part of the project were a probabilistic Grid workload generator and an extension to a popular distributed monitoring platform that enabled a more granular measurement of compute resource usage by the Grid applications.

A.1 GridLoader - Grid Load Generator

The following will present the work done on the Grid application simulator, called GridLoader. The motivation for developing such a tool will be outlined in Section A.1.1 while the requirement capture will be given in Section A.1.2. Section A.1.3 presents the implementation of the GridLoader, followed by the results of the functional and qualitative tests given in section A.1.4. Section A.1.5 concludes the GridLoader part of this chapter by summarising the findings and giving directions for further work.

A.1.1 Motivation

The simulation tools available in the Grid research community, as surveyed in Section 3.5, are helpful in studying various aspects of the Grid resource man-

agement and scheduling components before these are actually deployed. Once a solution is developed and installed on a testbed system, further testing is often needed to confirm proper end to end operation and integration with other components. At this point, a conflict exists between the need to subject the system to the conditions most closely resembling those found in the production environment, and the necessity to tune and control those conditions in order to facilitate system optimisation.

The motivation behind the GridLoader workload generator was to support in-site testing of the management components by creating a controllable application load with the job statistics similar to those experience in the production Grid environments. Such a tool would allow testing of the scheduling algorithm, monitoring components, and all other aspects of the SO-GRM management framework in a realistic usage scenario, without the problems usually associated with running on a live production Grid system.

A.1.2 Requirements

To represent a realistic Grid application, the GridLoader was required to simulate processor utilisation, memory allocation and network activity. The execution of the GridLoader would have to be fully parametrised, with a suitable tool to facilitate orchestrating large simulation runs. Such deployment tool would be used to decouple the overall statistical properties of the jobs submitted to a Grid cluster from the resource utilisation statistics of a single node.

One of the approaches for simulating a realistic application load, often used by benchmarking applications such as SPECmark [123], is executing a representative set of application code snippets in an automated way. This method gives a degree of repeatability [235], enabling comparison of hardware implementations by maintaining an unchanging application load. However, the probabilistic and self-organising nature of the SO-GRM components would require a more dynamic environment with a widely fluctuating load.

Another possible route for simulating realistic workloads is through a trace-replay system, such as the SimGrid for example (see Section 3.5.1). Although this is the most realistic representation of a production system workload, it may not be scalable to the desired length or utilisation fraction, it may be difficult to obtain, or it could cause the simulation to lock into specific properties of the system from which the trace was taken. Therefore, the aim with GridLoader was to be able to create a distribution of statistically similar loads while maintaining a level of ambiguity in order to challenge the self-organising and adaptive components.

An important requirement was to achieve the right balance between deterministic and probabilistic modes of operation. The simulation runs should be repeatable, and all simulation parameters should be adhered to if any incremental improvements to the management components are to be recognised. At the

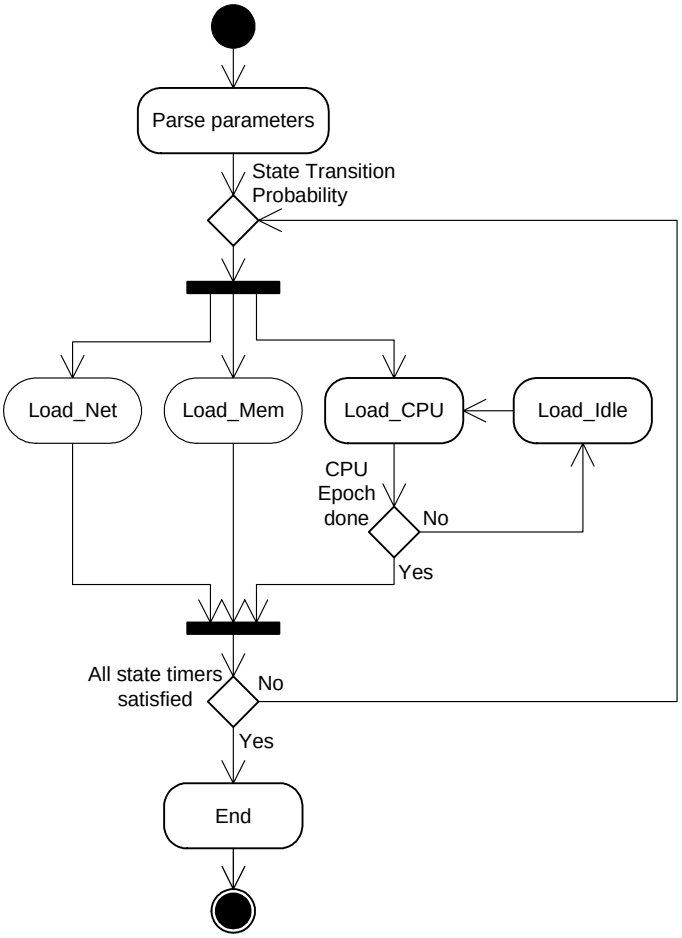


Figure A.1: Logical flow diagram of the GridLoader implementation showing the transitions between CPU, network and memory loading stages.

same time, a probabilistic element in the simulated application's behaviour is required for a realistic and diverse environment to form, and for SO-GRM component's adaptability and self-organisation to be exercised. Utilisation of different resources may also have to be simulated with a different distribution functions and parameters - network transfers may have substantially different statistics than the CPU utilisation.

The GridLoader application would need to be submitted through Grid middleware on the target site just like any other Grid application. To reduce source code compilation issues, a simple and portable code running under user privileges would be highly desirable.

A.1.3 Implementation

Following the established requirements, the GridLoader was implemented as a state machine, with different states representing CPU, memory and network loading stages. A logical flow diagram showing this structure is given in Figure A.1.3. State transition table can either be deterministic, moving through network

loading, memory allocation and CPU utilisation states in progression, or fully probabilistic.

Deterministic state transitions facilitate debugging of components under test, and creates a behaviour similar to an "embarrassingly parallel" [236] Grid application. A parameter sweep experiment is one common example of such an application: it stages the input data, allocates required memory and executes a CPU intensive core calculation that would usually produce a small result data set. The probabilistic state transition scenario leads to a more sophisticated model in which all three primary states are entered into many times with changing probabilities. Although this behaviour is more realistic, and representative of a more complex Grid application, it creates a very dynamic environment for all other components and possible faults are hard to locate and debug. This mode should be used in advanced stages of testing.

To ensure portability between Grid systems and the ability to compile and execute without administrator's influence, the GridLoader was written in ANSI C without any low level function calls or custom libraries. It was compiled successfully on Windows, Solaris and Linux platforms.

Application Simulation Stages

As previously shown on the logical flow diagram, the GridLoader has three states used to simulate the behaviour of a Grid application: network loading, memory allocation and intensive computation stages.

The network loading stage opens an UDP socket to an IP address specified as a command line parameter and transmits a random message 1400 bytes long for the duration of the requested network loading time. The inter-packet delay is parametrised at run time and is directly proportionate to the amount of bandwidth used. Once the timer signals the required time has passed, the socket is closed and a flag set for state transition.

Memory allocation state requests the kernel to increase the memory allocation to the process by the amount specified through a run-time parameter by using the *malloc* function call. UNIX memory management is handled very differently depending on the system implementation and the kernel optimisation options, and may prevent a user process from directly managing memory allocations. GridLoader ensures that the physical memory is actually allocated to the process by writing random data into the virtual memory space allocated by the kernel. The memory is freed during final clean-up state of the application, once all loading states have been completed.

Computationally intensive part of each Grid application is simulated in the CPU loading state. This state contains two real-time nested timers, one keeping track of the total amount of wall time spent in the CPU loading state, and one tracking short time slices in which CPU is toggled between full throttle utilisation

and idle. Very frequent swaps between these two stages result in a smoothed fluctuation of the CPU utilisation when observed at the sampling frequencies of less than 100Hz. Total wallclock duration of the CPU loading is specified at runtime, while the duration of each run-sleep cycle is determined in a random manner using a predefined probability distribution function. This function is randomly seeded at runtime, and partly parametrised through a command line option. The benefit of this approach is that even for equally parametrised runs, the actual CPU load trace would not be the same. This was an essential requirement for the testing of the I^3 security engine (see Section 2.4 and [42]): GridLoader was therefore able to simulate anomalies in the process behaviour and test the I^3 malicious process detection algorithm.

Once all the timers indicate that the requested loading metrics have been met, the final clean-up stage is entered in which the allocated memory is freed, network sockets closed, and a log file with details of the execution written. GridLoader can also operate in a debug mode which records detailed information about the state machine and the execution timers of each stage.

Parametrisation Options

All parameters of the GridLoader's simulation can be supplied either via the command line, or from a configuration file. Supported run-time parameters and their explanation is give in Table A.1.

To give the overall cluster loading a certain statistical property, and to facilitate the generation of the configuration files for larger GridLoader runs, an auxiliary application was developed in Matlab. Two types of parameters can be defined with either global or local scope. Global parameters influence the overall behaviour of the whole set of GridLoader jobs in a specific simulation run. These are used to coordinate the job set, and are detailed in Table A.2.

The variables defined in Table A.3 set the ranges for the generation of parameters influencing the behaviour of a single GridLoader instance on the node it is executing.

Deployment Scripts

The deployment application generates a file containing appropriate parameters for each GridLoader instance, and a configuration file for the batch scheduling script. The probabilistic nature of the GridLoader is here evident at different levels. At the global level, two job sets with the same parameters will not have the same values of individual local parameters, but in both cases those values will fit the same, requested, statistical distribution. At the level of a single GridLoader instance, two equally parametrised runs on the same machine will adhere to the parameters supplied, but will achieve those targets with a different resource utilisation profile.

Parameter	Description
NET	Total time for network transfer state, expressed in seconds
CPU	Total time of CPU loading state, expressed in seconds
MEM	Integer MBytes value of total physical memory to allocate
BURST	Inter-packet delay time, expressed in μseconds and used to control the amount of bandwidth used by the network transfer state
IP	Numerical IP address of the peer (or sink) for the network transfer state
PARETO_B	Pareto parameter B used to influence the idle time transitions in the CPU loading state. Large values of this parameter cause the long tail of the Pareto probability distribution to extend, leading to spikier CPU utilisation trace and larger average levels of CPU utilisation. Subsequent runs with the same value of parameter B will not produce equal traces due to different seeding values of the random number generator.

Table A.1: Description of the GridLoader command line parameters and explanation of their influence on the execution of a single GridLoader instance.

To help visualise the job set being run, deployment application produces a plot of parameter values with the relevant histograms, as shown in Figure A.2.

A.1.4 Self-Test Results

Before using the GridLoader to test other components of the SO-GRM management architecture, a test of its own reliability was undertaken. Primary concern was the quality of resource utilisation models and the adherence to the specified parameters such as the execution time and size of the allocated memory.

To test the reliability of the overall timekeeping, a set containing 120 jobs taking around 24 hours to complete was created and run in sequence on one of the nodes of the Grid testbed. A simple batch scheduler script was run on a "master" node and used to submit jobs through either the Globus Toolkit 2.2 middleware or the Secure Shell (SSH) to a set of dedicated "slave" nodes. Same job set was then re-run locally on the "slave" machines in order to differentiate between GridLoader's systematic error and any overheads introduces by the middleware. Figure A.3 shows a percentage difference between the expected and the actual execution times for a sample of 50 jobs and for all three different execution methods.

Running on the local node, actual the GridLoader execution times are less then 2% greater than expected. This is due to the system overheads such as

Parameter	Description
CPU_TOTAL_PARETO [A/B]	Defines the value of Pareto probability parameters for generating CPU loading times across the whole set of jobs. Any other standard probability distribution function could be used with appropriate parameters.
ITERATIONS	The number of GridLoader jobs to create
NEXTREQ [MIN/MAX]	Used in a simple batch scheduling script, defines the range of wait times before submitting the next job. The values are normally distributed within the set range.
NEXT_HOST [MIN/MAX/PREFIX]	Also used in simple batch scheduling operation, defines the next host's IP address to which the job will be submitted.

Table A.2: Description of the parameters used by the MATLAB deployment script and influencing the global behaviour of a number of GridLoader instances run as part of one experiment.

setting up the network transfers, allocating the memory and random number generation, which are not accounted for in the timekeeping of the program. As this level of increase in the execution time is intrinsic to the operating system, and would be present for all the applications, we found that a realistic and accurate simulation of the total length of the job can be achieved using GridLoader.

As previously described, a loose control on the level and shape of the CPU loading can be exercised by specifying different values of the Pareto parameter B at run time. A parameter sweep test was undertaken to establish the upper and lower bounds of these values that provide a usable result. During these tests it was noted that a low value of the parameter will result in a longer duration of CPU idle time, and thus a lower average load. Higher values of the shaping parameter cause Pareto probability function to return high values for the duration of the CPU intensive loops and thus lead to a higher average utilisation and pronounced load spikes. GridLoader's probabilistic routines will create a similar, but not equal, trace for each equally parametrised run.

Reliability of the duration of the network transfers was established as part of the overall test of the GridLoader timekeeping. The influence of inter-packet delay parameter was examined through a parameter sweep test. By using network monitoring package Iperf*, the bandwidth utilisation between the "slave" node executing GridLoader and a designated traffic sink node was measured. The inter-packet delay parameter provides a soft control of the amount of bandwidth used, and not a strict upper or lower limit. This kind of probabilistic behaviour is sufficient for the required simulation of the network traffic and, considering the

*see http://dast.nlanr.net/Projects/Iperf/

Parameter	Description
CPU_LOAD_PARETO_B [MIN/MAX]	Sets the upper and lower bounds on the Pareto B parameter; range of values is generated using normal PDF.
IP [LOW/HIGH/PREFIX]	Defines the range of IP values for the target IP address of the GridLoader network peer. Could be defined as a single IP address to simulate a master-slave Grid environment.
MEM [MEAN/MIN]	Sets the GridLoader's memory allocation parameter. The value is calculated by adding a random number with the mean of MEM_MEAN to the minimum value defined in MEM_MIN.
NET [MEAN/MIN]	Sets the GridLoader's network transfer time parameter. Calculated in the same way as the memory value above.
BURST [MEAN/MIN]	Sets the GridLoader's inter-packet delay parameter. Calculated in the same way as the memory value above.

Table A.3: Description of the parameters used by the MATLAB deployment script and influencing the local behaviour of each of the GridLoader instances run as part of one experiment.

aims of the simulation, its probabilistic nature is beneficial. The use of the UDP network protocol, and its lack of bandwidth control mechanisms, could lead to network congestion issues in large GridLoader simulation runs. It remains to be assessed whether such conditions would impair the running of the simulation or add another realistic aspect of the production network environment.

Sequential memory allocation and freeing has been monitored using the Ganglia system, as shown in Figure A.4. The tests were carried out to confirm the actual physical memory is being allocated, and that this could lead to memory contention as is the case in the production environments. The granularity of the allocations is one megabyte but could easily be reduced.

A.1.5 Conclusions

GridLoader provides a way for parametrised and probabilistic simulation of application CPU, memory and network usage. Deployment scripts facilitate creation of run-time parameters for large simulation runs, enabling these to follow statistics of jobs observed on the production Grid facilities. Testing of the GridLoader functionally and reliability has been undertaken and reported on.

During the stand-alone testing phase of the GridLoader, a number of minor problems and issues were discovered.

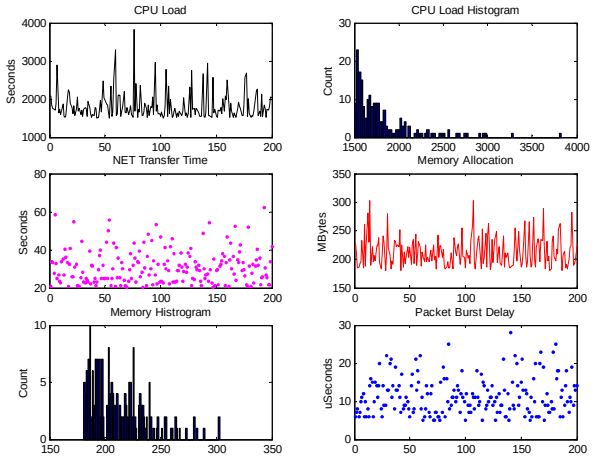


Figure A.2: Distribution of individual parameter values for a sample GridLoader experiment consisting of 200 jobs.

From the implementation perspective, a better CPU loading algorithm would prove very useful. Some cases exist where a constant, predefined level of CPU load should be simulated, such as in visualisation applications or other applications bound not computationally but by some other factor. These could not be precisely simulated using the currently implemented probabilistic approach.

GridLoader heavily depends on the quality of the random numbers generated within the programme, and the seeding mechanism for the random number generator. Although better generators than the one used in GridLoader are available, these would require additional libraries which may not readily be available on the target platforms. As no adverse effects associated with random number generations were observed during debug runs, the current approach is considered adequate.

Numerous problems were caused by the real-time clock resolution and the

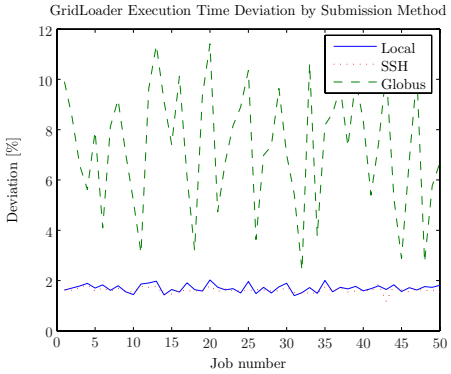


Figure A.3: Reliability testing of GridLoader job execution time plots a discrepancy between requested and achieved job runtime depending on the job submission method used.

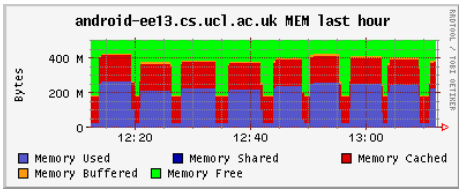


Figure A.4: Reliability testing of GridLoader memory utilisation showing allocation of and de-allocation of physical memory.

lack of synchronisation between the Grid nodes. Globus X.509 certificates have an associated validity period with a one second granularity, and in a network without proper clock synchronisation a certificate may become valid on one machine before it does so on another. This leads to the job being rejected due to the incorrect credentials, an error message often associated with other issues within the Globus Security Infrastructure and Certification Authority problems.

Overall, the parameter generator application and the GridLoader were successful in creating a job set with given statistics, and executing it according to the parameters required. Appropriately parametrised GridLoader will be able to simulate a realistic Grid application workload and present a diverse and varied load to the Grid management components on test.

Development of the GridLoader is a distinct contribution of this thesis. Apart from its primary intended use as a Grid application simulator described above, GridLoader can potentially be used as a testing tool for confirming end-to-end application level operation of Grid middleware. With a suitable parameter set, the GridLoader could also be used to stress Grid hardware and middleware components to the edge of their operational envelope, thus exposing any possible points of failure or performance bottlenecks.

A.2 Monitoring Framework

An extension of the widely used Ganglia Monitoring Suite [160] has been developed to provide an enhanced monitoring capability for jobs running on the Grid, and support the long term collection and storage of their resource utilisation traces. This section will present the motivation for this work, system requirements, implementation details and the results of the functionality and reliability tests before concluding with some final remarks and directions for further work.

A.2.1 Motivation

Current Grid monitoring systems, as previously summarised in Section 3.4, offer a scalable and effective monitoring of resource utilisation on a per-node basis. As one must assume a general case where Grid nodes will be used by other (system or user) applications, these measurements are not representative of the actual

resources used by any single application. Even in the case of a dedicated Grid host, the footprint of the current Grid middleware, management and security components is such that the overall node resource utilisation will be very different to that of a single user application.

The author's motivation was to extend one of the current monitoring systems to provide process-specific measurements of resource utilisation in an unobtrusive and scalable way. Extension to an already established monitoring system would have the benefit of an already established user base, giving access to a wider source of data. It will also remove any switching cost from the user's perspective and alleviate administrator's reservations about installing an unproven piece of software.

A.2.2 Requirements

The basic requirements for a Grid monitoring system are support for a wide range of operating systems and hardware architectures, effective data storage methods, and the use of efficient and standardised communication protocols. An extensible metric sampling interface, the possibility of integration with the Globus MDS, and the support for XML encoded messages were the additional requirements for a successful integration with other SO-GRM management components.

The monitoring system of choice should be able to integrate per-process resource utilisation metrics into the standard flow of measurement data, and fully support storing and retrieving of such additional information through its usual data access methods.

A.2.3 Implementation

After surveying the monitoring tools available, the decision was made to base the extended monitoring framework on the Ganglia cluster monitoring system [160]. Ganglia was selected for its extensible data collection interface, effective storage of data in a fixed size round-robin databases, the use of XML encoded measurements, and customisable unicast and multicast delivery protocols. It has previously been extensively used with Globus Toolkit and successfully integrated with the MDS using the Glue Schema [237]. Various platform-specific information providers have been developed, and this modular design offers a clear path for the implementation of per-process resource utilisation monitoring.

Ganglia Functionality

The monitoring suite is implemented through a set of Ganglia applications, compiled code, and shell scripts developed by the author. All code was written with portability in mind and relies on UNIX standard libraries and script commands. Figure A.5 presents the layout of the monitoring components in a block diagram. Ganglia Cluster Monitoring core provides two daemon modules:

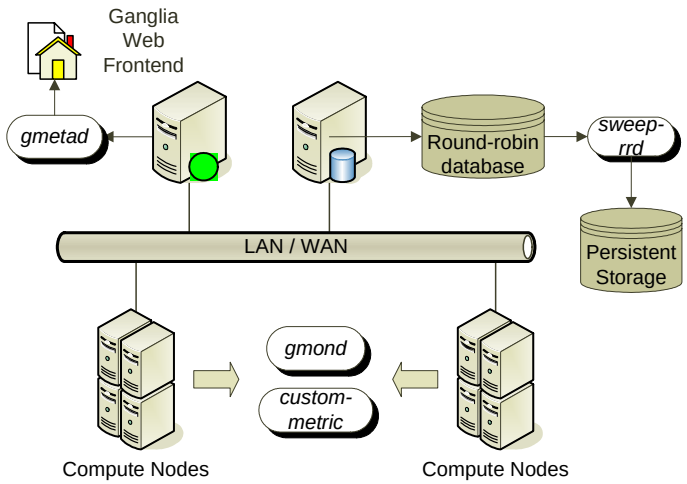


Figure A.5: Block diagram of Ganglia monitoring components integrated with author's custom metric providers.

- *Ganglia Monitoring Daemon (gmond):* collecting basic information about each node in predefined time intervals, encoding it in XML and providing the network transport mechanism.
- *Ganglia Meta Daemon (gmetad):* receiving the information broadcasted by all or some of the monitoring daemons, and storing it in the round-robin databases. It also answers queries about overall state of the cluster, and provides a programmatic interface to the queries on the data contained in the databases.

Round-robin database (RRD)* is a fixed sized database targeted at storing the time-series data. Each database can contain several data sources (DS), and each data source has a number of round robin archives (RRA). These archives could be thought of as a set of differently sized and stacked gears, with each cog slot containing one sampled value. On database creation the frequency of rotation of each of these gears is defined, and a consolidation function (CF) is given for each data source. Once the gear makes a full turn all of its data is passed through the consolidation function (usually average, minimum or maximum) and the result is written as one sample point in the cog of the higher hierarchical gear. The size of the database is kept constant, since the high frequency data is kept for a limited duration before being consolidated. Depending on the target application, this behaviour may be a desirable feature or a disadvantage.

Ganglia Monitoring Daemon can use either unicast or broadcast UDP packets to transport the XML encoded measurements. Each *gmond* daemon can be set up to either listen to other daemons (mute mode), transmit its measurements to other peers (deaf mode), or do both. By configuring certain nodes to be muted or deafened, a resilient distributed system can be created. In our test

*see http://oss.oetiker.ch/rrdtool/

implementation, all but one Ganglia monitoring daemons were configured in deaf mode. One node in the network run the non-deaf daemon, as well as *gmetad* daemon, and provided storage for all databases. This centralised network configuration was appropriate provided the size of the test network (no more than 10 nodes at any time), and the goal of the tests.

Information Providers

The author has developed custom information providers to monitor the CPU utilisation and memory footprint of each process submitted through the Grid middleware. These were implemented either as a shell script (using a UNIX standard *ps* command), or as a pre-compiled application using the *libgtop* library. Functionality is similar, as both implementations run as a daemon on each Grid node and periodically sample the CPU and memory utilisation. Criteria for process selection, and the information collected, are fully customisable. The monitored processes can be selected by their identifier (PID), executable name, or by username under whose credentials they are running. Information reported can include any metric available through the UNIX */proc* system.

Although process selection based on the PID is the most efficient and unambiguous method, current implementation of the Globus Toolkit (V3) does not pass the PID of the remote process to the job scheduler, nor does it make this information available through MDS or any other means. This is a widely recognised implementation issue, impeding improvements in several areas such as grid job workflow management and scheduling concurrency. Next versions of the Globus Toolkit should address this problem. Once the per-process monitoring data is collected, it is transmitted either using Ganglia's *gmetric* shell command or by using Ganglia's API libraries, depending on the implementation.

Database Management Tools

The characterisation of the Grid workload data presented in Chapter 4 depended on the availability of an extensive amount of high frequency monitoring data from a representative Grid cluster. Although alternative data collection options were subsequently made available, for workload characterisation studies the consolidation feature of the round-robin databases was not beneficial as the highest resolution measurements would be quickly lost through averaging. A shell script, *sweeprrd* in Figure A.5, was developed to perform an automated data extraction from the RRD databases. The script can be configured to retrieve data on specific nodes and specific metrics of those nodes, or collect all the data available.

Time stamped measurement values are formatted in a comma delimited format, and stored as a flat text file. The script can either run as a daemon process or be invoked by the UNIX standard *cron* scheduling daemon. The frequency of execution is customisable with the obvious lower limit of at least one sweep within

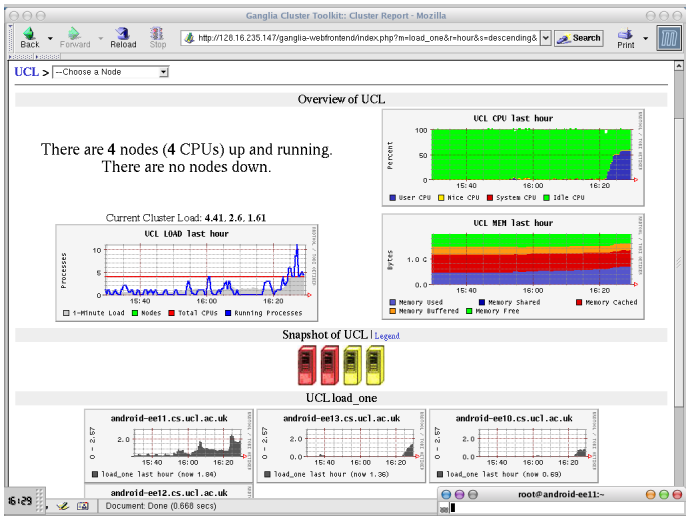


Figure A.6: Cluster level screenshot of Ganglia monitoring web interface

the duration of the shortest round robin archive in the database (to prevent any data being lost through consolidation). Database sweeps can be invoked as often as necessary and at any time; the script will only extract new samples from the RRD database and append them to an already present output file.

A.2.4 Test results

First phase of the monitoring suite tests was aimed at confirming the proper installation and the basic functionality of the Ganglia suite. After modifications to Ganglia's default settings, it was necessary to ensure core functionality has not been affected and stable operation was maintained. Ganglia version 2.6 was deployed on both BT and UCL administrative domains of our testbed Grid. Figure A.6 shows a typical screenshot of Ganglia web front-end displaying overview of hosts in the UCL domain.

In the second phase of testing, per-process monitoring components were introduced and observations were made on the stability of the system, quality and reliability of the measurements, and any increase in the system resources utilisation. Screenshot in Figure A.7 shows a single monitored node in the Grid under heavy utilisation, while screen detail in A.8 shows *globus-cpu-utilisation* metric, revealing the CPU utilisation attributed to a single Globus submitted job.

The third phase of the tests was designed to establish the overall monitoring functionality and the quality of measurements. A sample GridLoader set containing 50 jobs with Pareto distributed execution time was run on a single machine on the Grid testbed. A full set of metrics including Globus-attributed and total CPU load were recorded through the monitoring suite with one second resolution, averaged and published over 15 second periods. Jobs were submitted from one of the machines in the cluster to a different machine in the same cluster using an appropriate Globus command. A simple master-slave scheduling was used, iterat-

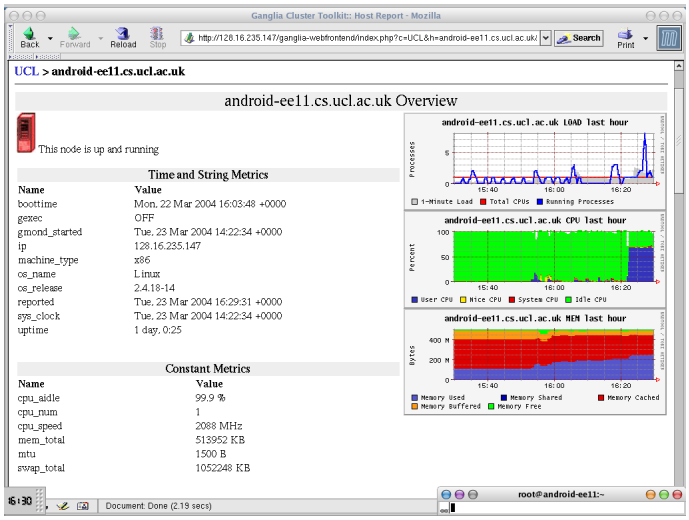


Figure A.7: Node level screenshot of Ganglia monitoring web interface

ing through the job list and allowing 45 seconds between the job completion and next job submission for any transient machine loading to settle. These transient loads were created by the Globus toolkit job completion procedures such as the results stage-out, process cleanup and accounting file updates.

The measurements revealed the difference between the GridLoader generated load and the total system load which includes various background processes associated with the Globus middleware, kernel time servicing network transfers, memory allocation and process scheduling. The differences were most obvious at the start and the end times of each job, while the machine loading is high, but the CPU time is not yet attributed to the process being submitted.

This experimental data has also exposed a peculiar behaviour of the process monitoring component which leads to a ramp-up effect in the observed loading

Figure A.8: Process level screenshot of Ganglia monitoring web interface

measurements. This low-pass effect causes large variations between the total node utilisation value and the Globus attributed CPU load at the beginning of job execution. The software routine responsible for collecting those measurements uses the UNIX standard process reporting calls, and these return CPU usage as a decaying time average since process initiation [238]. To improve the accuracy of measurements, a version using kernel "jiffies" [239] was made, but this improvement results in the loss of portability between platforms.

Most of these issues where in the local monitoring component. Regardless, successful overall operation of the system was confirmed, and sampled data was correctly integrated in the Ganglia data handling flow (including Web-based data visualisation). Data extraction tools operated effectively and reliably with no lost or duplicated samples. Data obtained was readily analysable, and had immediately provided insight into the extent of difference between perceived and actual resource usage by Grid processes.

Resource footprint of the monitoring system was acceptable (estimated at less than 1% of CPU time); although an increase was noted as the number of processes to be monitored grew. This is attributed to the computationally expensive parsing of the processes table required to obtain process IDs of the monitored jobs, and depends strongly on the criteria used for selecting the monitored processes.

A.2.5 Conclusions

Presented monitoring solution addresses the problem of obtaining per-application resource usage statistics on Grid cluster nodes and provides a solution for the whole monitoring cycle, from measurement data collection, to visualisation and extraction for off-line analysis. The system has been developed on an open framework to support programmatic access to the data by other Grid management components. Implementation has taken into account expressed reservations of the cluster administrators to running third party compiled daemons on their networks, and has developed a transparent monitoring system based on a widely used monitoring application. The chosen approach scales well, being based on a proven core and complemented with the maintenance scripts designed to facilitate deployment and management. This solution seamlessly integrates measurements specific to the needs of the advanced scheduler research within an established monitoring framework. Off-line data analysis is facilitated with the use of the data extraction scripts developed.

Appendix B

Additional Workload Characterisation

The prediction of job execution times based on the historical information, one of the distinct contributions of this thesis, used the methods rooted in the observations made in the analysis of a representable Grid workload presented in Chapter 4. This workload study applied the exploratory data analysis[169] (EDA) techniques to suggest the causes of the observed phenomena and to support the selection of appropriate statistical tools and techniques that can be used to effectively "mine" the data for previously unknown and potentially useful information.

The fallacy of the EDA approach is that a systematic bias is often present due to the erroneous approach of using the same data set to both suggest and verify certain hypotheses. This problem can be avoided by cross-validating the hypotheses on a collection of independent confirmation samples.

The purpose of this chapter is to perform such validation by using an alternative Grid workload trace. This will offer supporting evidence to the findings of the workload characterisation given previously in Chapter 4 and confirm, to the extent possible, that observed phenomena are indeed universal to the Grid workload. In doing so, this chapter will also further validate the job execution time forecasting approach taken and ensure its applicability in a range of Grid usage scenarios.

B.1 The Workload

As the Grid technology is relatively new, few truly large-scale, multi-purpose, production Grid environments have been deployed. Those facilities that are operating do so under strict security and data protection rules making it very difficult to obtain, analyse and publish work based on their usage statistics. This

is especially challenging for studies, such as this one, requiring highly granular, job- and process-level data for which specific user permission must be granted.

Apart from the UCL's CCC Grid cluster workload, the author has managed to acquire another job trace from a member cluster of one of the largest European Grid operators compromising more than 200 sites and over 30,000 CPUs. The trace does not contain the full set of job properties, so the following analysis will focus on the job inter-arrival process and execution time - two key aspects from the job runtime prediction point of view.

Access to this data was given subject to the identity of the Grid project and the site in question remaining undisclosed.

B.2 General Workload Properties

The workload compromises of almost a quarter of a million jobs submitted in the nine month period between August 2004 and May 2005. During this period, about 3.5% of jobs have executed for less than one second, the resolution of the accounting file clock, and are deemed to have failed on runtime. This failure ratio is consistent with the CCC findings and those reported by others.

The distribution of active users, VOs and job names indicate that a large number of users belonging to very few VOs have submitted almost all the jobs using very few job names. Such scenario is an indication of the unfortunate administrative policy at the site encouraging submission of jobs with generic names and failing to introduce transparency in the mapping of Grid users to local credentials.

Calculated application efficiency of 83% is very high, and in line with the CCC findings, re-affirming the view that currently run Grid applications are compute-bound. The overall cluster utilisation of 22% is low compared to the CCC but on par with other academic and dedicated commercial Grids. The summary of these workload properties is given in Table B.1.

B.2.1 Job Inter-arrival time

Figure B.1 describes the job arrival process at this facility by plotting the run sequence plot of the job inter-arrival times and their cumulative distribution function. Around 30% of the jobs arrive in batches with less than one second apart compared to almost 80% of such quick succession job arrivals present in the CCC. Such difference could most likely be attributed to a significantly lower overall utilisation of this facility and the appropriately longer periods of time without any incoming jobs. Regardless, bursty job submissions are still an important feature of the arrivals distribution.

The remainder of the inter-arrival times distribution seem almost linear on the log scale and this is further affirmed in the normal probability plots shown

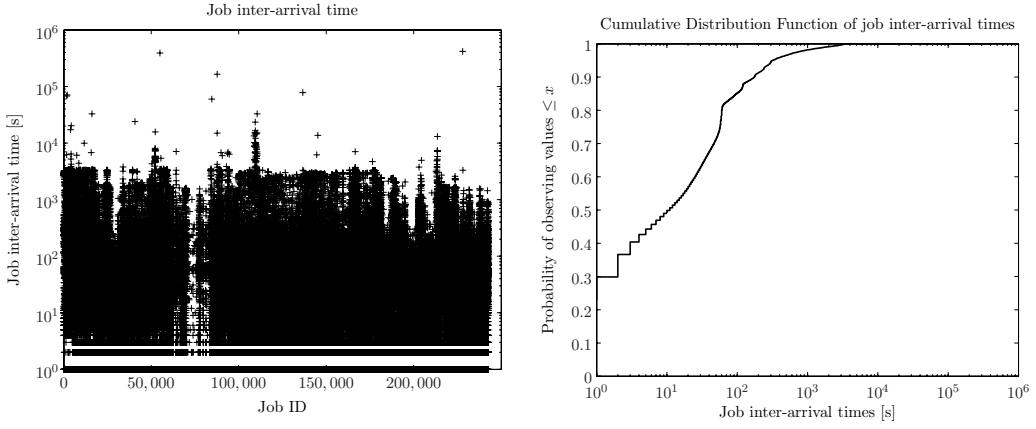
First job time	14.08.2004 22:36
Last job time	11.05.2005 14:07
Number of days	270
Worker nodes (CPUs)	70(140)
Number of recorded jobs	242,695
Failed (0 sec) jobs	8,618
Unique users	56
Unique VOs	8
Unique job names	12
Total job wallclock time	705,566,432s (8,166 days)
Total job CPU time	585,289,080s (6,774 days)
Mean Cluster Utilisation	22%
Mean Application Efficiency	83%

Table B.1: The summary of the workload analysed

in Figure B.2. The linearly scaled plot exhibits very strong skew towards smaller values, while the logarithmically scaled one shows very good linearity for values larger than one second. The inter-arrival times of this facility, provided batch submissions are treated differently, could be modelled using a log-normal distribution.

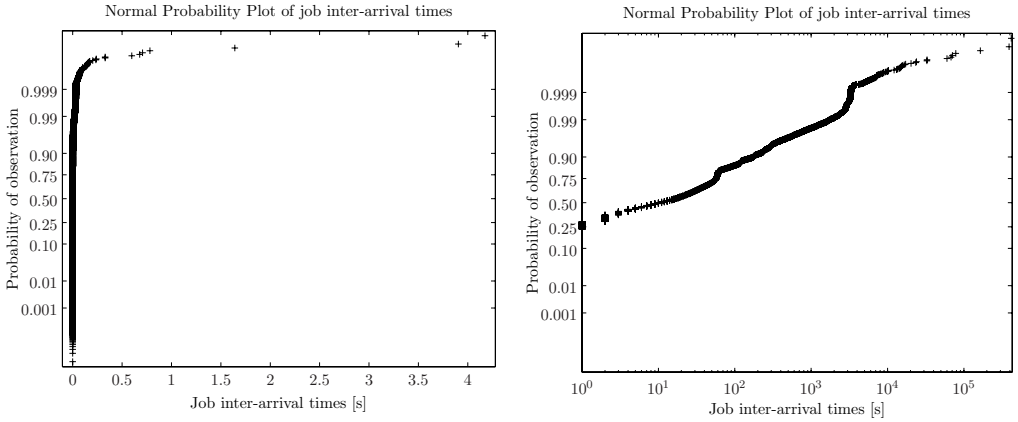
The cyclic pattern and the seasonal variations of the job submission process was an important characteristic of the CCC workload and has also been found in the usage statistics at this facility. Figure B.3 shows the total number of submitted jobs in each month of the trace, for each date in the month, day of the week and hour of the day.

The monthly plot, which runs from August to May of the following year, clearly shows a ramp-up effect at the beginning of the facility production life followed by a steady fluctuation of job submissions and a tail-off towards the end



(a) Job inter-arrival time (run-sequence) (b) Job inter-arrival time (CDF)

Figure B.1: Run-sequence and CDF plots of Job Inter-arrival times



(a) Linear (b) Logarithmic

Figure B.2: Job inter-arrival times normal probability plot

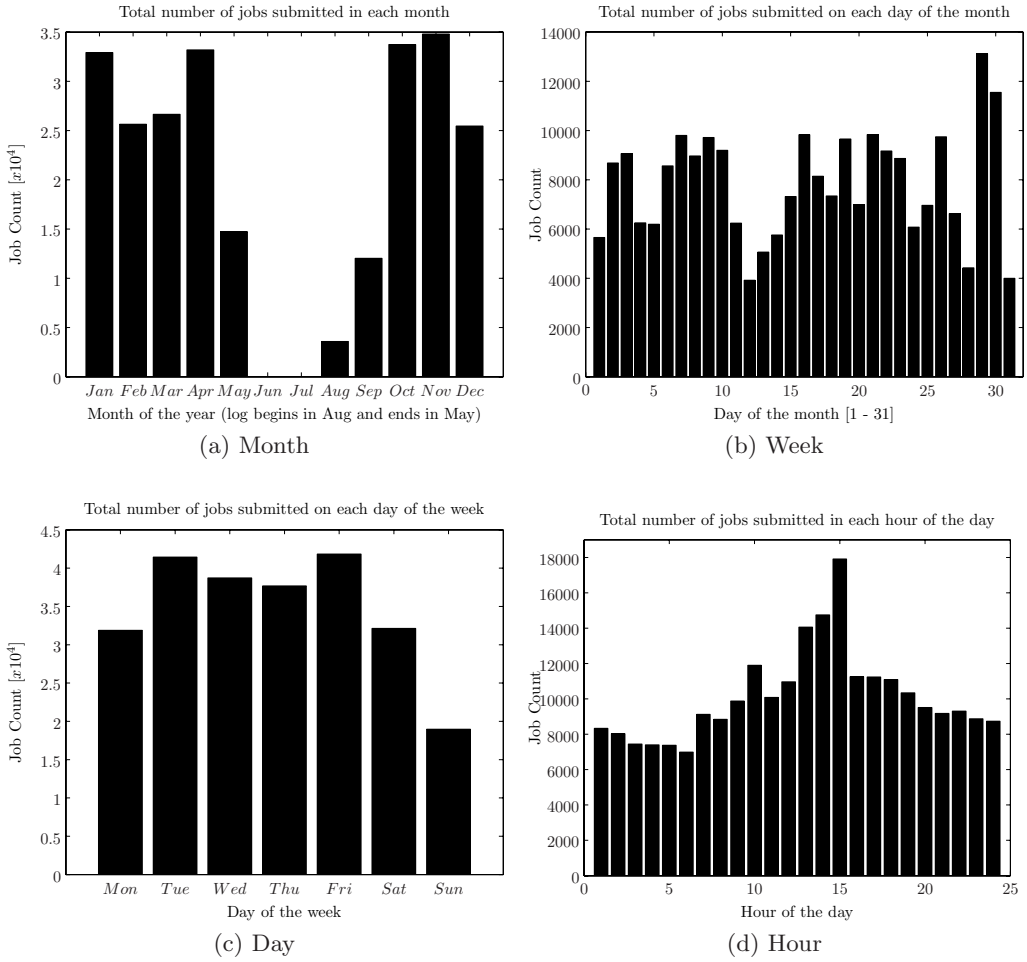
of the workload trace. Again, the plot for the dates of the month does not reveal much as it is strongly dominated by the weekly job submission pattern.

This facility exhibits a slightly different but still comparable pattern to that of the CCC. Both Grid clusters see the lowest number of submission on Monday and Sunday, but in this facility's case the Wednesday peak is replaced by a more spread out distribution between Tuesday and Friday, with Saturday also seeing a high number of job submissions.

The hourly distribution of job arrivals is similar to the one seen at the CCC with peaks in the late morning and early afternoon followed by a steady stream of jobs throughout the night. The smaller peak at around 10am, presumably for jobs which will finish before the day's end, is followed by a larger peak at 3pm which would probably see jobs running overnight or longer being submitted.

The presence of any long-tail behaviour in the distribution of job inter-arrival times has been assessed by using the complementary cumulative distribution function plot shown in Figure B.4. The plot shows that for values of inter-arrival times larger than 10 seconds, the tail of the distribution follows the fitted Pareto function very well over an extended range of almost five orders of magnitude. Similarly to the behaviour observed at the CCC, the distribution of inter-arrival times is long-tailed at this facility as well.

Finally, the self-similar nature of the job arrival process was tested by estimating the value of the Hurst parameter using the rescaled range analysis on the job inter-arrival times. Figure B.5 shows the resulting plot which exhibits good linearity and indicates a Hurst value of 0.81. This is 0.04 lower than the value indicated for the CCC but is still a very strong indication of a self-similar process.



(a) Month (b) Week

(c) Day (d) Hour

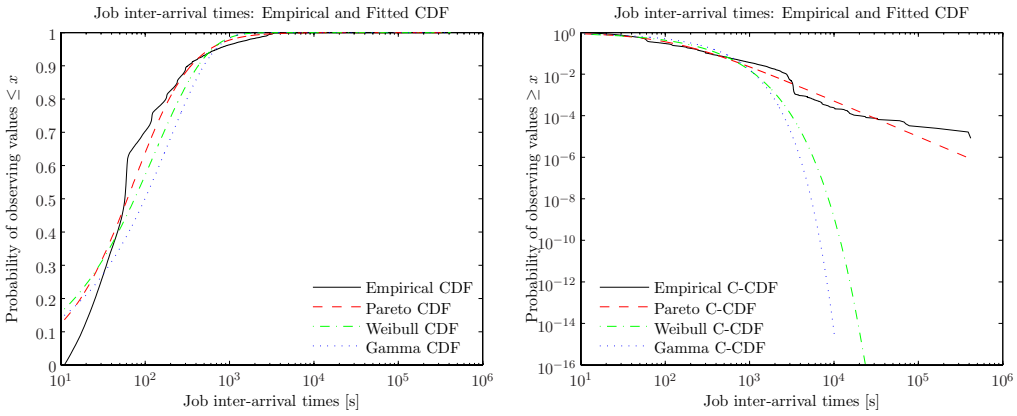
Figure B.3: Job submission count: cyclic behaviour

B.2.2 Wallclock Execution Time

The run-sequence plot and the cumulative distribution function of job wallclock execution times is shown in Figure B.6. Compared to the CCC job runtime distribution, this facility has a higher fraction of shorter running jobs and a lower percentage of longer running ones.

Common to both Grid facilities is the absence of any prominent modes or predominant values of job runtimes. Analysing the normal probability plot shown in Figure B.7 it is clear that apart from some skew for runtime values of more than 10,000 seconds, the distribution is a very good fit to a log-normal one. Each runtime value is as probable as any other throughout this wide range, thus making the process of fitting a sensible forecast model to such data set difficult if not impossible. These findings further support the need to section the workload into more predictable partitions before applying selected forecasting methods.

Figure B.8 plots the total wallclock execution time of jobs as a function of their submission time. Due to the shortness of the workload trace, and the variability of the job runtimes, plots showing monthly and date of the month fluctuations



(a) 10sec+ tail CDF (b) 10sec+ tail CCDF

Figure B.4: Job inter-arrival times: long-tailedness and representative functions

do not offer much insight into the usage pattern of the facility.

The weekly usage cycle plot reveals that by far the longest running jobs are submitted on Fridays, with the ones submitted on Mondays, Saturdays and Sundays having the shortest runtimes and mid-week jobs falling in between. Such usage scenario is very similar to the one observed on the CCC Grid and an evidence of users self-prioritising their work.

When analysed together with the job submission cycle shown in Figure B.3(d), the hourly job runtime pattern reveals the tendency of users to submit shorter running jobs in the morning, anticipating their completion in the afternoon, and longer running jobs in the late afternoon and early evening hours which run overnight. Indeed, the 10am, 3pm and 6pm peaks of job submission seen in Figure B.3 correspond to the peaks in the job runtime lengths. This is another example of the human perception of time and the corresponding modality and seasonality in the expectations of job services times.

Workload characterisation of the CCC Grid cluster has indicated that the

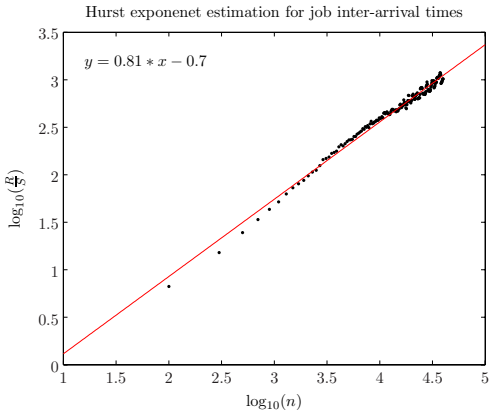
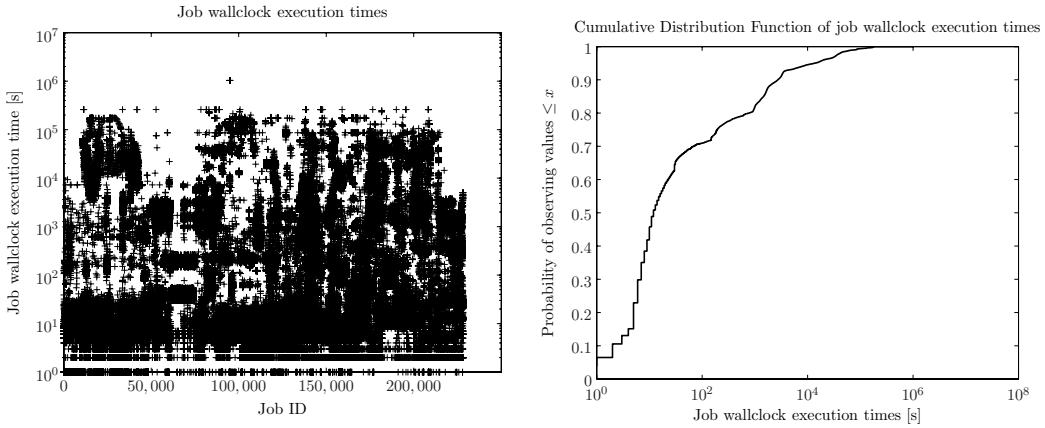


Figure B.5: Job inter-arrival times Hurst exponent estimation using the rescaled range (R/S) method

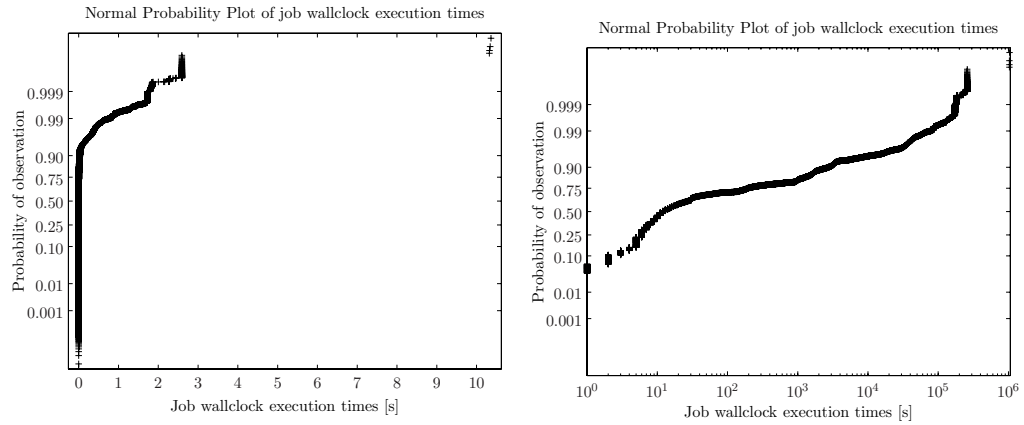


(a) Job wallclock execution time (run-sequence) (b) Job wallclock execution time (CDF)

Figure B.6: Run-sequence and CDF plots of job wallclock execution time

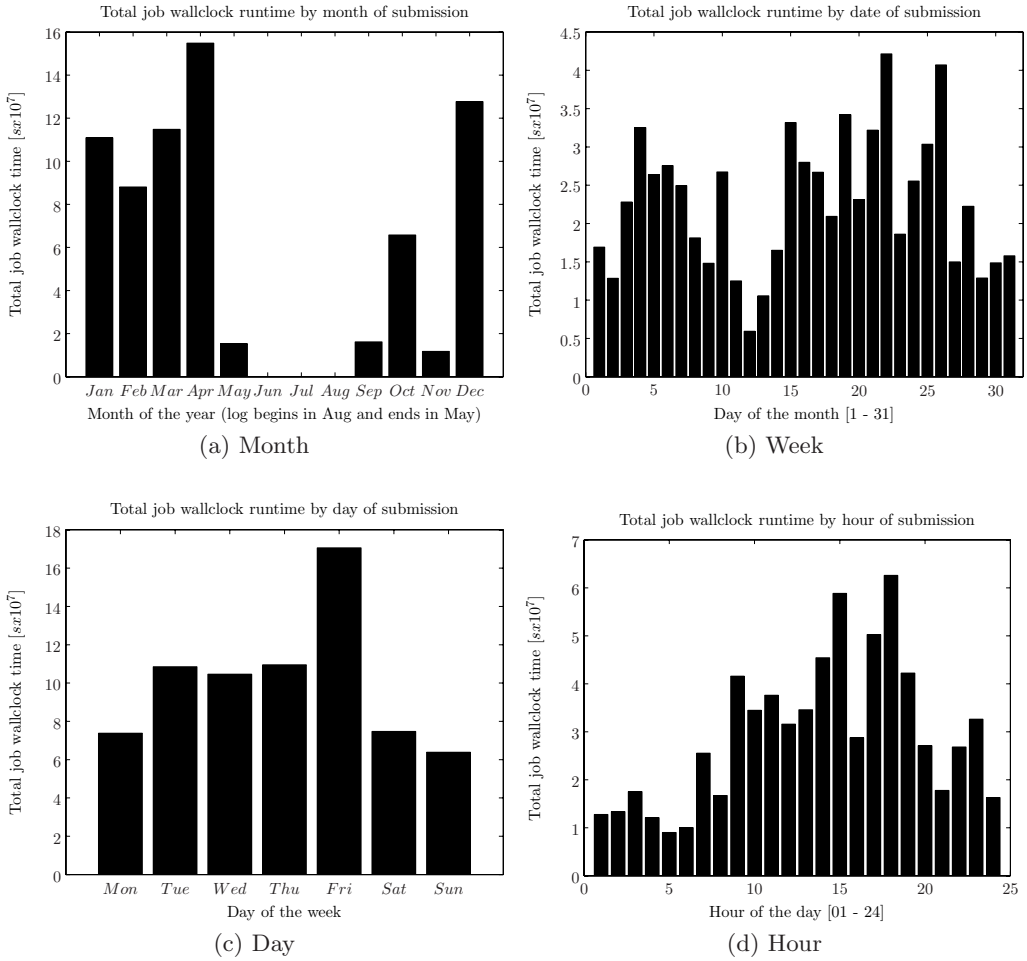
distribution of job runtimes exhibits a strong long-tailed behaviour. The result of a similar test done on this facility, plotted in Figure B.9, shows a good fit to the Pareto model up until around 10,000 seconds. The following steep and modal decline in the probability of observing values higher than 10^5 is most likely attributed to an upper bound in the running time of submitted jobs, a "kill time", which is often enforces in high-performance compute facilities. Unfortunately, the author could not establish whether such a policy applied in the case of the analysed facility.

The self-similar properties of the job runtimes were estimated using a rescaled range method. The fitted line in Figure B.10 estimates the Hurst parameter, with good linearity, at 0.80 which is an indication of a strongly self-similar process. Considering that the Hurst parameter of the CCC job runtimes was 0.87 it can be concluded that Grid runtimes do have a self-similar nature.



(a) Linear (b) Logarithmic

Figure B.7: Job wallclock execution time normal probability plot



(a) Month (b) Week

(c) Day (d) Hour

Figure B.8: Job wallclock runtime: cyclic behaviour

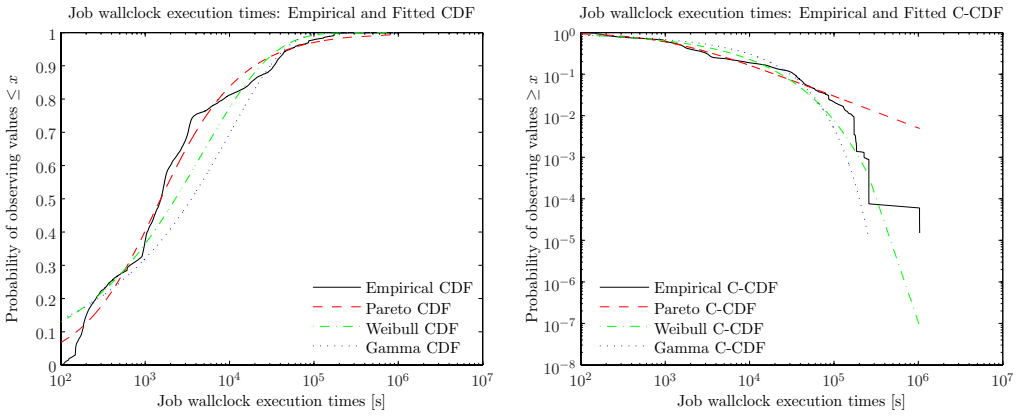
B.3 Meta Differentiation and Workload Diversity

Once the need for partitioning the workload into clusters of jobs with the similar statistical properties, "behaviour" or greater predictability, the question arises how could these pivot partitioning metrics be defined. This thesis has proposed using a mix of job meta-data and temporal properties to reduce the variability of the job runtime distribution. The effect that such job partitioning would have on the location and dispersion of runtime values will be examined in this section.

B.3.1 Job runtime v. job meta-data

The usage statistics available for this facility contained the anonymised identification of the user, VO and the job name being submitted, as well as the queue to which the job was sent. Figure B.11 uses box-plots to show the difference in the distribution of job runtimes with respect to the four pieces of available job meta-data.

Plotting the distribution of each user's job runtimes, as seen in Figure B.11(a),



(a) 100sec+ tail CDF (b) 100sec+ tail CCDF

Figure B.9: Job execution times: long-tailedness and representative functions

shows how vastly different their statistics are. Although some users do submit jobs with a very large inter-quartile range, the majority runs jobs with a much smaller dispersion. Since these users are assigned to very few VOs, partitioning based on the owner VO in most cases returns unsatisfactory results. The job names have even less resolution, as only three are commonly used.

The final plot shows the correlation between the job runtime and the queue to which the user has submitted the job. As the CCC had only one queue, such statistic was not available, but the findings by other researchers on the lack of correlation between the implicit user predictions of job execution time (expressed by queue selection) and the actual job runtime were often noted in this thesis. In the case of this facility, it is clear that such findings are accurate. While the *Test* and *Short* queues do have lower medians and inter-quartile ranges than the remaining ones, the dispersion of the job runtimes in the *Long*, *Day* and *Infinite* queues is almost identical and the median value is decreasing instead of increasing. The *Batch* queue has seen almost no job submissions. A Spearman's

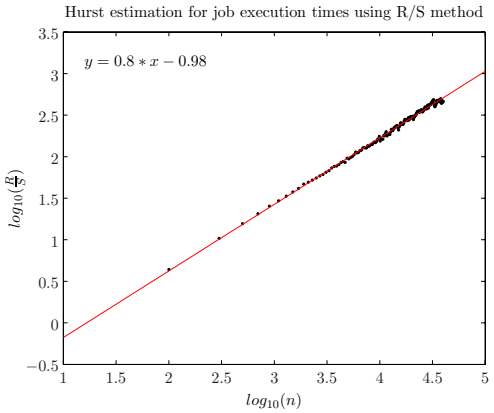
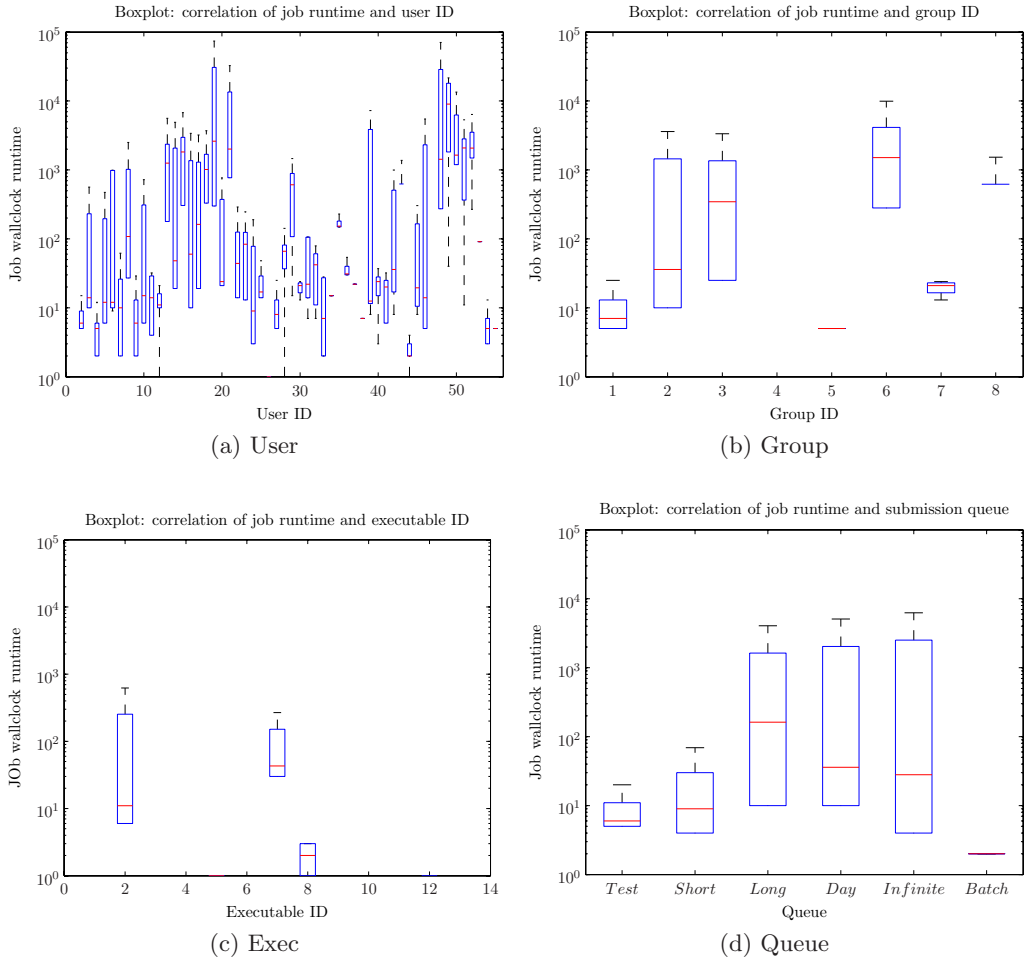


Figure B.10: Job wallclock execution times Hurst exponent estimation using the rescaled range (R/S) method



(a) User (b) Group

(c) Exec (d) Queue

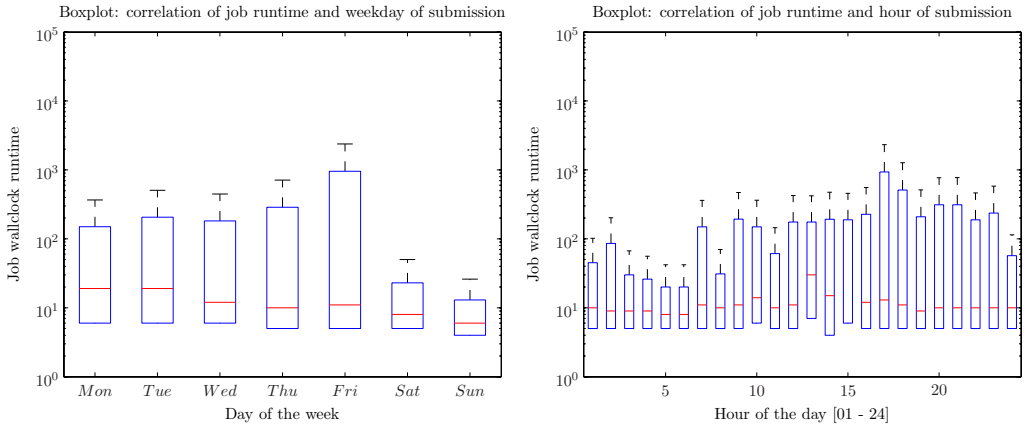
Figure B.11: Job wallclock runtime correlation: meta-data

rank correlation coefficient between the job runtime and the queue selected was 0.28 indicating a very small positive correlation. Such findings reiterate the problem of relying on the user estimates of the job execution time and further motivate the need for an autonomous and automated prediction system.

B.3.2 Job runtime v. job submission time

One of the novel aspects of this thesis was in using the temporal job properties to partition the workload into more closely related groups. Such approach makes use of the observations that Friday jobs run longer, that jobs submitted in the late afternoon tend to execute throughout the night, or that job runs that are closer in time tend to be more strongly autocorrelated.

Figure B.12 shows the central tendency and the distribution of job runtimes according to the day of the week, or the hour of the day, in which they were submitted. The plots show a steady rise in the execution lengths throughout the week with a peak on Friday, followed by much shorter execution times at the weekend. The hourly plot reveals a similar pattern with the longest running jobs



(a) Day of the week (b) Hour of the day

Figure B.12: Job wallclock runtime correlation: temporal data

submitted at late afternoon, and a distinctly different profile of execution times during the working day and overnight.

When applied as a sole partitioning criteria, this observed correlation between the job's submission time and its execution time may not yield results as good as the application of clustering based on the job meta-data. Its real potential however is in further differentiating these meta-data based partitions according to workflow habits of a specific users or Virtual Organisations.

B.4 Conclusions

The purpose of this appendix was to present the workload characterisation of an additional multi-purpose, production Grid facility, which would support the findings of the CCC usage study presented in Chapter 4, and the subsequent methods of predicting job execution times given in Chapter 5.

By focusing on the job arrival process and the wallclock duration of the job execution, the analysis has found substantial similarities between the two Grid workloads. Both of the studied properties have a log-normal distribution, long-tails and are significantly self-similar. There were also strong cyclic patterns on the weekly and daily scales.

The potential of the temporal- and meta-based job partitioning in reducing data variability (and thus increasing predictability) was confirmed with the submitting user, the time and the day of job submission identified as key pivot metrics. It was also shown that the user's selection of the queue to which the job will be submitted is a poor indication of how long such job will run for.

Considering the difficulties of obtaining representative Grid workloads, the results presented in this chapter provide strong support to the conclusions drawn from the analysis of the CCC workload, and further justify the methods and

approaches used in the forecasting of job runtimes based on the historical information.

Appendix C

Commercial Aspects

The following appendix will examine the commercial value of the presented PhD work, discuss possible ways of commercialising researched approaches, methods and techniques, and investigate feasible scenarios for monetising added value offered by the autonomous deadline scheduling on the Grid.

This additional work was kindly sponsored by the joint collaboration of University College London and London Business School through the Centre for Scientific Excellence*, established in 2000 to promote entrepreneurship within the fields of science and technology. The author is grateful for their ongoing support.

C.1 Grid Computing Technology

Among several definitions of Grid Computing, from a business perspective the most applicable one defines it as a collection of computing and storage elements running a layer of software (called middleware) which is presenting these resources as a unified platform. Grid resources can be geographically distributed, within different administrative domains and running on various supported hardware and software, but through a Grid middleware layer these are all presented as a unified Grid service.

Grid computing is in many ways a potentially disruptive technology. By enabling concentration of compute power away from the end user, and by offering it as a metered service on a pay-per-use basis, it opens up a new market segment of computational power providing. It creates a new business model focused on competitively selling Grid services in an open market, by suppliers who are leveraging economies of scale in hardware procurement, management cost and operating expenses.

*http://www.cselondon.com

C.2 Business Potential of Grid Computing

Grid technology can potentially offer great cost savings and increased productivity to businesses in a wide range of compute intensive industries such as engineering, finance, automotive and biochemical. Deployed at the core of a company's computing environment the Grid can bring the following benefits:

- **Reduced Total Cost of Ownership** through a unified and centralised management interface that reduces the running costs through economies of scale.
- **Linear capacity growth and capital expenditure** as hardware can be added to the Grid in smaller, more granular steps, rather than investing in large server farm upgrades.
- **Increased utilisation** through resource virtualisation and formation of a universal utility platform with no hard partitioning of resource.
- **Highly adaptable and agile computing platform** as a variable and dynamically adaptable amount of resources can be used to deliver each service thus helping to align available resources with current business priorities.

Deployed across the company's general computing capital, such as employee workstation and terminals, the Grid can be used as a"cycle scavenging" platform to run computational jobs on underutilised computers and thus extract more value from the investments already made.

C.3 Grid Computing Value Chain

Computational grids are effectively a large and distributed computer clusters, found in academia and industry requiring powerful, high-throughput facilities. These large institutions have established relationship with equipment manufacturers and vendors, are often tied in with a long term contract, or have funding commitments related to a specific supplier. Majority of these high performance clusters were made to order, using low volume or specialised hardware, and upgraded throughout their long life-cycle.

In this environment, switching costs are very high and supplier lock-in is strong. The Grid could significantly disrupt hardware supplier's power as it enables high-performance, high-availability clusters to be assembled out of commercial off-the-shelf components (COTS). This has the effect of shifting significant value extraction potential from hardware manufacturers to middleware vendors and system integrators.

Following is a brief explanation of key links in the Grid value chain and major companies competing in each segment.

C.3.1 Hardware Manufacturers and Suppliers

Companies at the beginning of the chain are traditionally hardware manufacturers and suppliers with strong focus on business IT sector, competencies in large server deployments, and experience supporting mission critical hardware.

A major profit share of these companies comes from high value contracts to supply their top of the range enterprise hardware to large institutions. This revenue stream was disrupted 2001 to 2004 by a slowdown in corporate IT spending and businesses focusing on getting value for money.

Grid technology is unsettling to these large hardware manufacturers as it reduces their product differentiation: just about any hardware component can be used to create a Grid cluster and the Grid middleware will enable jobs to be executed quickly and reliably. As a consequence, hardware manufacturers are trying to add more value to their enterprise level hardware and differentiate it better from their low level kit (usually by adding management, deployment and monitoring tools). A range of hardware is now also labeled as Grid-enabled, a property which still has no universal meaning and is mostly used for marketing purposes.

Major hardware manufacturers with keen interest in supporting and developing the Grid concept are IBM, Sun Microsystems, Hewlett-Packard and Dell Computers.

C.3.2 Middleware and Software Vendors

With the introduction of computational clusters made of COTS components, and with big steps in virtualisation and interoperability of heterogeneous kit, the middleware (or software glue) that enables their interoperability and management is becoming a more important components of the overall system.

Companies in this part of the value chain are based on the software developer or retailer business model with valuable income coming from the support and customisation contracts. The competition in this sector is limited, and most middleware vendors are operating in their own niche market segments. Product development cycle is long and based on a major early adopter whose custom solution was generalised to cover their entire industry. The companies in this part of the value chain are growing quickly and have to be learning as they go along. Human capital and up to date skills are very important, leading to expensive labour force.

Although gross extracted value at this point in the chain is less than at the hardware manufacturing level, the profit margins are higher, the business much less capital expenditure intensive and client lock-in still very strong. Essential at this stage are strong links with both hardware manufacturers (to ensure compatibility and as a sales channel to undecided clients who are just entering the Grid

market) as well as high value clients (who may have specific customisation needs and can serve as valuable references).

The largest companies at this value chain level are recent privately held start-ups, spun off by academics involved in the Grid research, or people with the specific knowledge of the technology who were previously with one of the big hardware manufacturers. These include Platform Computing, Avaki, United Devices and to some extent Sun Microsystems.

C.3.3 System Integrators and Consultants

Popularisation of Grid computing has increased the need for knowledgeable system integrators and consultants to guide a new Grid adopter through the selection of appropriate hardware, Grid middleware and Grid enabled business applications.

As with so many new technologies, the Grid has been suffering from compatibility issues, difficult and time consuming deployment scenarios, and high levels of ongoing management and maintenance of the early systems. The experience of people who have already been through this process is invaluable and a very good basis for a professional services business model.

The companies in this value chain segment are mostly small start-ups or consulting businesses with looser or tighter connections to a larger Grid hardware manufacturer or middleware vendor. Some of their founders come from academia while some are ex-project managers from hardware manufacturers or early adopters of the Grid technology. These Grid consultants command high profit margins, but are dependant on the number of new and repeat clients. Good relationship with all parties in the value chain is therefore essential. Some of the currently better known consulting firms are Globus Consulting and Platform Computing.

C.4 Probabilistic Deadline Scheduling

A job scheduler is an important part of the Grid middleware whose task is to order the jobs waiting to be executed in such a way that the utilisation of the system (or some other given metric) is maximised. The waiting queue can have thousands of jobs and there may be hundreds of machines on which these jobs can run and the scheduling process quickly becomes a complex optimisation problem.

C.4.1 The Need for Better Scheduling

The performance of the scheduler influences the throughput of the whole Grid cluster, user's satisfaction with the computing service they are getting, and the profitable use of Grid operator's resources. Currently deployed schedulers employ a range of modified first-come-firs-served (FCFS) batch approaches. This

means that the jobs are executed in the order in which they arrived, unless some administrative policy explicitly favours jobs from a certain user or group. This static prioritisation is of poor selectivity and leads to low levels of resource utilisation. It also does not match the human workflow often based on the notion of job deadlines.

Job schedulers that were developed specifically for academic use usually do not deliver in the commercial sector. Better Grid schedulers able to fit human workflow through the use of deadlines, offer quantifiable Quality of Service (QoS), and be more easily manageable are clearly needed.

Development of this next generation of schedulers depend on the ability of the Grid middleware to forecast the execution time of jobs in the queue, their future arrival rate and the presence of any cycles or patterns in the workload. The research work undertaken as part of this PhD thesis offers a way of obtaining those kinds of information from the statistical models based on the historical job execution data.

C.4.2 Probabilistic Deadline Scheduling Proposition

The methodology described in this thesis enables automated forecasts of job execution times based on the historical models of previous job runs. The approach uses additional information associated with the job (such as submitting user, Virtual Organisation (VO), date and time, application name etc.) to look for usage cycles, patterns and correlations which reduce the variability of the data and increase the accuracy of predictions.

The technology used enables several important improvements in Grid scheduling and Grid resource management:

- **Analyse usage patterns and workload distribution.** A workload model is developed by monitoring and analysing the jobs submitted to the Grid. This model is then used to analyse usage patterns of individual users, VOs or periods of the day or week.
- **Estimate execution time of a job.** By using a model of execution times developed for a certain user, executable or execution scenario, it is possible to predict how long a newly submitted job will run and establish a margin of error for such predictions.
- **Detect and track out-of-ordinary job characteristics.** Continuous observations of the state of the Grid and the running jobs enables the system to spot sudden and significant changes of job characteristics. This information is then used to ensure quality of scheduling and if necessary bring this behaviour to the attention of system administrators.

These core abilities enable new functionality and offer added value to the process of Grid resource monitoring, management, provisioning and job scheduling:

- **Support for deadline scheduling.** A predictive, probabilistic scheduler is able to offer users execution of their jobs to a certain deadline. Knowing how long a job will run enables the scheduler to re-arrange the job queue out of order and maximise the likelihood of jobs completing by their deadline. For example a short job with a long deadline would be moved further back in the queue to free up resources for a job whose deadline is tighter, regardless of the order in which they were submitted.

- **Increased overall system usage.** Together with a resource pricing system, probabilistic scheduling would enable users to trade off their "computing budget" against the urgency of their work. A job with a relaxed deadline, or one submitted at off-peak hours, would cost less to process then an urgent job run at peak times. This tried and tested yield management approach evens out usage distribution throughout the service period and lowers peak to average resource requirement ratio.

- **Dynamically align resource use with corporate priorities.** As deadline is specified on a per job basis as a measure of each job's priority, hard partitioning of resources can be avoided. Provided resources are available, a relaxed deadline job from a high priority user would not block an urgent job from a lower priority user. In this way maximum flexibility and fairness to all users can be maintained while aligning resource use with business priorities.

- **Provides business intelligence on computing usage patterns.** User's workflow and habits, usage patterns and job execution scenarios are revealed through detailed monitoring of resource usage and correlations between jobs and their "softer" properties such as submitting user, Virtual Organisation, time or command line parameters. This valuable insight can help in system planning and provisioning, spot problematic applications or users, and reduce hotspots and congestion on the computing platform.

The benefits of a predictive deadline scheduling approach to an enterprise running a large Grid cluster serving numerous users with widely varying resource requirements can be significant. Ways of capitalising on those benefits and the presented technology are discussed in the next chapter.

C.5 Possible Exploitation Routes

Assuming that scientific validity, practicality and fitness for scheduling purpose of the probabilistic scheduling approach presented in this thesis is confirmed, several exploitation routes are open. In this section a range of possible commercialisation options will be discussed, and their benefits and problems analysed.

C.5.1 Patenting

As with many other scientific discoveries, patenting is the first and foremost opportunity of generating revenues. A possible commercialisation route for the author's research would be to patent a method of making execution time forecasts based on the histories of previous runs, the use of time-series analysis for making such forecasts, and the integration of pattern matching and outlier detections to help improve the quality of predictions.

While obtaining a patent is never easy or straightforward, in this case further complications arise from the fact that it is a mathematical or logical construct implemented in software that needs to be patent protected. This has traditionally been hard to do and companies have previously resorted to implementing software in specific hardware to qualify for an "aparatus" as required by some patent authorities. The European Union has been considering legislation on software patenting from as early as 1999 but has always come against a very strong opposition from software manufacturers and users alike. At the time of writing the EU has begun third round of consultations on the software patents but it seems unlikely swift or clear action will be taken on this issue any time soon.

Apart from evident problems and legal challenges in patenting a software invention, the application procedure itself is a lengthy and expensive process. Provided a patent is granted, it then must be upheld in the face of challenges from competitors and defended from infringements. Since patent litigation can be very costly, a large company infringing on a small firm's patent can prolong the process and financially weaken the competitor.

The revenue model in a patenting business is a straightforward collection of royalties. The pricing structure depends on the strength of patent protection, added value that the patented solutions delivers to the main product, and the cost to the licensee of developing a similar technology while not infringing the patent. The benefits of the intellectual property licensing model are modest capital investment requirements and ongoing costs directly related to the level of its research and development effort.

All things considered, intellectual property licensing approach can grow a profitable and sound business, but must rely on very strong patent protection and bespoke leadership in a given market segment.

C.5.2 Third-party Scheduler Add-on

Examples abound in the software marketplace of smaller companies developing add-on solutions that significantly improve the usability, performance or functionality of a larger applications. This model could be used to develop a probabilistic scheduling add-on for the scheduling systems already deployed on the production computational Grids.

By relying on an industry accepted scheduler, and developing only an execution time prediction module, the amount of initial development and coding work would be minimised. This also means a shorter time to market and a lower seed investment would be required. Entering the market by improving an already existing scheduler leverages its installed user base, and significantly reduces user switching cost as changing their middleware provider would not be necessary. With low barriers for entry, this approach could lead to a high conversion factor if the add-on becomes an accepted "standard" upgrade in the industry and may tempt a buy-out by the company behind the actual scheduler.

The success of the business based on this model depends on the management of the product development cycle, prudent cash flow control and a timely hiring of effective marketing and sales force. Once the initial product has been developed, product margins can be high if the distribution channel and the customer support expenses are well managed.

A major problem with this commercialisation route is that in a bid to lock in the customers, few commercial middleware providers make their schedulers based on open standards and published interfaces to which an independent add-on could be attached. Since the performance of the overall scheduling system greatly depends on the core scheduler over which we would have no control, problems, poor overall performance or reliability issues with the system could be brought into connection with our scheduling add-on and affect negatively on the start-ups reputation.

Most importantly, unless patenting the predictive elements of the add-on is possible and could offer strong IP protection, large scheduling system providers could move to integrate similar technology in the new versions of their products. With this in mind, possibly the best exit strategy with this approach would be to position the company as a likely buy-out target by an established Grid scheduling software developer.

C.5.3 Standalone Probabilistic Scheduler

By deciding to take full control of the job scheduling in computational Grids and use the apparent benefits of probabilistic scheduling, a possible commercialisation route would be to develop, sell and support a fully fledged standalone Grid scheduler. This approach would offer the flexibility to implement all the insight and research done for this PhD thesis but would also expose the start-up company to a great amount of risk.

The development and testing of a mission critical component such as a scheduler would be very costly and time consuming. It would certainly require expert management and a strong, knowledgeable programming team. Attracting employees of this profile would be hard for a small start-up company, and would most probably involve equity sharing remuneration packages.

A new entrant to the Grid middleware market would face high barriers due to the market's monolithic nature, supplier lock-in and informal supplier selection methods based on previous references, experience and perceived reputation. The company would have to build their own client base (whose switching costs would be high), and help them through the migration process (involving a high volume of expensive support time).

With a completely independent scheduling solution the commercialisation venture could certainly capture more value than as an add-on provider but at the cost of much greater capital investment, longer time to market and profitability, and significantly greater risk. This business model would require a strong strategic partner, a well funded company willing to move into the computational Grid market and looking for a new technology to break ground. The level of financial support extended to the start-up would certainly influence the equity distribution between shareholders and may yield a relatively modest return for the entrepreneur.

C.5.4 Professional Services - Consulting Business

With years spent researching the Grid scheduling, user behaviour patterns, and Grid technology and middleware, a reasonable commercialisation of the author's know-how would be a consulting role in a professional services business. The probabilistic scheduling method and its associated job runtime prediction software could serve as a bespoke tool that, coupled with an in-depth analysis of client's requirements, can deliver significant added value to their computational Grids.

This business model would offer more than a scheduling system add-on, it would provide a customised scheduling, tuned to client's specific requirements. It would require sizable initial investment in order to move the predictive algorithms from academic test bench into production environments but would not require extensive support or sales network. The model could offer good profit margins and a rewarding working environment for the entrepreneur. If a foothold in the market was established, additional consulting work could be achieved through horizontal expansion into other Grid related fields.

The most important factor for success of this business model would be client acquisition. The very labour intensive nature of customised approaches limits the possible client pool to large organisations with expensive or specialised equipment whose high utilisation is essential, or organisations running mission critical applications requiring very specific scheduling. The expense of developing a custom solution would only make financial sense in these cases.

The barriers for entry would be high: with no previous track record a solid proof that the predictive technology works, and that the start-up has enough know-how to apply it, would be required. The consultancy would have to develop

a unique and recognisable approach to distinguish itself from competitors and imitators. In such environment the start-up would depend strongly on finding its first client, an early adopter willing to try out a new approach.

The ongoing success of the company would mostly be influenced by its recruiting strategy and its ability to attract capable and knowledgeable consultants, perhaps through an equity sharing plan. Structured management from as early on as possible would be needed to help the founder delegate responsibility and allow the company to grow.

C.5.5 Overview

Considering different commercialisation options in the context of a new business start-up, the most important factors are the amount of seed capital required, the assessment of the business's profit potential and the amount of time it would take to develop a marketable product or service. The overview of those important aspects for proposed commercialisation routes is given in Table C.1.

	CapEx	Profitability	Time to market
IP Licensing	○	○	◐
Scheduler Add-on	◐	◐	◐
Standalone Scheduler	●	●	○
Professional Services	○	◐	●

Table C.1: Overview of commercialisation options available with respect to their required level of capital expenditure, anticipated profitability potential and required time to market.

The balance between the risk and the reward is subject to the investor's personal circumstances and the expectations of the industry as a whole. Given this overview, the following section will discuss in further detail the chosen commercialisation route and the justification for such decision.

C.6 Selected Approach - Scheduler Add-on

After considering all four possible commercialisation aspects given in the previous section, developing a predictive scheduling add-on for an already deployed scheduler offers the best balance between the potential profits and the amount of risk a start-up could commit to.

While patenting the predictive scheduling approach plays an important role in all business models considered, legal obstacles and the burden of proving novelty to the patent authorities would make a successful patent application very hard. Software companies in similar situations usually prefer to retain the know-how and seize the opportunities of market innovators capitalising before imitators are

able to catch up. A start-up is unlikely to have sufficient financing available to reach the market fast enough.

After further discussions with colleagues who have managed larger software development project before, it became clear that developing a fully featured scheduler, of adequate reliability to be used in the large and often mission critical production environments would be prohibitively expensive for a small start-up. This option remains open if a large strategic partner is found, and its expertise used to speed up such development. Even if such opportunity arises at some later point in time, work done on developing a scheduler add-on would not go to waste and it would certainly serve as a proof of concept and of company's ability.

Finally, running a professional services business based on the custom Grid workload analysis tool may not be sufficient to sustain profitability and growth. The question of author's experience and that of related academics who would be involved may also prove an issue with future clients. While there is presently a growing need for outsourced Grid knowledge, it is likely that this trend will continue, and from the aspect of offering Grid consulting services the author can only benefit by gaining further experience.

The following sections will examine the strategic and financial aspects of launching a new business around a scheduler add-on based on the predictive scheduling technology.

C.6.1 Strategic Analysis

Assuming a company will be set up to commercialise on this research work, it will certainly have very limited resources. A focused strategy and well researched market environment in which it will operate will help it create a competitive edge over similar new ventures. This section will outline such company's primary objectives, its biggest advantages over its competitors, a strategy for bringing a new product to the market, and breaking into profitability in about three years time.

Mission Statement

The company's primary aim is to enable clients a more productive use of their computational Grid infrastructure. This would be done by developing a job scheduler supporting executions to a user specified deadline, and by offering clients novel tools to analyse, plan and provision their Grid usage.

Core Competencies

The core competency of the company is in its in-depth research of Grid usage scenarios, workloads, job traces, job meta-data, and user behaviour. A secondary competency is the tool-set and the know-how to statistically analyse this data

for a large number of patterns and correlations that can help reduce the amount of uncertainty in the dataset.

These competencies can be applied to a wide variety of Grid related products and services and can contribute significantly to end-product value. As they present accumulated knowledge, it would be hard for competitors to quickly or easily imitate them.

Competitive Advantage

The primary competitive advantage of the company is a product differentiation one. Our product will deliver benefits to the clients (such as scheduling to a deadline) exceeding those offered by the competitors. This will influence the positioning of the firm in the market, both in fending off low-cost competition and conquering the markets of other, feature-rich, scheduler.

Target Scope	Advantage	
	Low Cost	Product Uniqueness
Broad (Industry Wide)	Cost Leadership Strategy	Differentiation Strategy
Narrow (Market Segment)	Focus Strategy (low cost)	Focus Strategy (differentiation)

Table C.2: Porter's generic strategies table identifies three possible strategies (cost leadership, differentiation and focus) depending on the firm's application of their main advantages (cost advantage and differentiation) in either broad or narrow scope.

According to Porter's generic strategies presented in Table C.2, the company would be pursuing a focus (differentiation) strategy due to its product uniqueness and narrow target market scope. By focusing in closely on its niche market, the company can enjoy a high degree of customer loyalty and thus raise entry barriers for direct competitors. As a downside, their narrow market focus increases buyer power and makes them vulnerable to acquisition by broad-market competitors or large customers.

SWOT Analysis

SWOT (strengths, weaknesses, opportunities, threats) analysis offers and insight into internal and external environment in which the company will operate. It plays an important role in formulating overall strategy and in matching the company's resources and capabilities to the competitive marketplace in which it operates.

- **Strengths:** The following resources and capabilities will be the basis for developing a competitive advantage

  - Proprietary know-how in the analysis of Grid utilisation, usage patterns recognition and the use of social factors for better Grid usage modelling
  - Cost advantages from utilising work already done as part of the PhD research
  - People capital and networking with relevant contacts in the Grid industry and academia
  - Ability to adapt to market conditions or specific client needs

- **Weaknesses:** The absence of certain strengths may weaken the ability to deliver on company strategic goal
  - No patent protection for the core predictive technology
  - No established brand or reputation
  - Lack of access to the key distribution channels

- **Opportunities:** The market environment in which the company operates holds key opportunities which can be developed into revenues
  - Large client interest in a potentially disruptive technology
  - Unfulfilled customer need for a scheduling method well suited to their workflow
  - Dynamic market with large growth potential

- **Threats:** Critical actions or changes in the external environment which can present threats to the company and jeopardise execution of the business plan
  - Failure to produce a reliable and efficient product
  - Move by the current Grid scheduling makers to integrate similar functionality into their core products
  - Emergence of substitute or competing products
  - Shifts in the cluster technologies, IT spending or high performance computing strategies away from the distributed approaches and the Grid computing

Since the company is a new start-up business, its opportunity cost is low and risk tolerance high. With a new and exciting product in the development, it should follow a strength-opportunities (S-O) strategy which would see it pursuing opportunities that are a good fit to its strengths.

Porter's Five Forces Analysis

Michael Porter's Competitive Advantage [240] provides a well known "five forces" model for the industry analysis based on pure competition. It is helpful in understanding the market conditions the new company will encounter and focuses the management process on possible problems and company's strengths that can be leveraged to overcome them.

- **Barriers to entry** - *Strong*
  - The patents and the proprietary know-how needed to develop the sophisticated Grid scheduling and resource management components
  - New entrants require specific assets (mostly appropriate human capital) to enter
  - High brand loyalty and high switching costs.
  - Restricted access to the distribution channels and clients.
- **Threats of Substitutes** - *Medium*
  - Dangers of substitute technologies making effective Grid scheduling obsolete:
    - departure from distributed or utility computing concepts
    - stronger affirmation of high-end workstations
    - monolithic parallel computers or a significant jump in the computing power of single chips reducing the need for computational Grids.
  - Industry adopting and / or standardising on one of the other alternative Grid scheduling approaches.
- **Supplier Power** - *Low*
  - Product mass-production is standardised (software duplication).
  - Product R&D depends to an extent on the highly skilled workforce, but with no strong labour union and with good availability on the labour markets.
  - Backward integration threat by purchasers is considerable; possible acquisition by a Grid middleware developer looking to extend its scheduling product portfolio.
- **Buyer Power** - *Strong to Medium*
  - Concentrated buyers; few large institutions and enterprises have computational Grids, even fewer require sophisticated scheduling methods.

  - Large buyers will purchases significant proportion of the company's software licenses.
  - Significant buyer switching costs once on our scheduler lowers buyer power.
  - Scheduling is also a critical portion of Grid middleware further lowering buyer power.

- **Degree of Rivalry** - *Low to Medium*

  - A small number of firms developing the Grid middleware and the scheduling software reduces rivalry.
  - Strong market growth reduces rivalry by leaving plenty of space for all competitors.
  - Low fixed costs usually experienced by the software industry reduce rivalry.
  - High switching costs lead to lower levels of rivalry.
  - High levels of product differentiation (schedulers are developed to fulfil a specific need no other scheduler on the market does) reduces rivalry.
  - Since buyers are concentrated and hard to switch, strategic stakes are high - a company can either lose market position or experience great gains leading to intensified rivalry.
  - Being a global technology trend, the Grid computing attracts a diversity of rivals from different cultures and market philosophies creating a volatile and intensive rivalry.
  - Industry shakeout is possible due to the strong market Growth and a disbalance in the capital strength of the rivals.

The above overview of the Grid middleware and the scheduling software industry indicates a lucrative market with a strong growth potential, and a low to medium rivalry intensity. With a low level of supplier power, and a threat of substitutes mostly dependant on the long term acceptance of the Grid technology, the risk seems to be well balanced. The high entry barrier is significantly reduced by the work already carried out as part of the doctoral research, and offers the author a good starting position compared to other potential market entrants.

C.6.2 Financial Analysis

Providing preliminary financial analysis of the profitability, cash requirements and financing structure of the start-up enables the entrepreneur and potential investors to judge the merits of the business, and whether it meets their risk requirements and anticipated rate of return. The following takes a look at the

financial potential of the company and analyses the cash flow anticipated in the first three years of operation.

Financial Potential

The financial potential of the start-up business will influence its valuation, its attractiveness to the potential investor, as well as the amount of risk he or she is willing to take. It is influenced by the following factors:

- Cluster, utility and Grid computing market capitalisation and growth rate
- Proportion of the market attributed to the sales of the middleware and scheduling software
- Price of those scheduling components, which would influence the retail price of our scheduling add-on
- Market capture of our scheduling add-on
- Our overall profit margin

Table C.6.2 outlines the profitability scenario based on currently available market data. The analysis assumes a steady grow in the market capitalisation of the Grid IT sector and a percentage increase in the spend share of the grid middleware (due to increasingly commoditised hardware). The number of shipped scheduler units is hard to judge based on the available data and supplier pricing is usually negotiated together with a consultancy or support contract. The stated figures are thought to be reasonable estimates and a conservative projected growth was used. Our market capture was initially estimated at around 5% climbing to 20% in year 3 with a very modest increase in the base price of the scheduling add-on. The projected revenue in year 3 was therefore estimated at £1.35 million.

C.6.3 Cash Flow Analysis

Currently, the product is in the proof-of-concept stage. To successfully bring the product to the market, the company must be able to sustain itself on seed funding until it begins to generate profits. Prudent cash management during that period is essential, and good estimates of the start equity required are a basis of this.

First Year Operation

The following assumptions have been made when estimating the cost of operations in the first year.

Development costs: It is estimated that a five strong software development team would need ten to twelve months to deliver the first stable, marketable

	Year 1	Year 2	Year 3
Grid, cluster and utility computing market cap.	£1,000,000,000	£1,200,000,000	£1,500,000,000
Grid middleware percentage	10%	15%	20%
Grid middleware market cap.	£100,000,000	£180,000,000	£300,000,000
Schedulers shipped	3,500.00	4,000.00	4,500.00
Scheduler avg. price	£8,000	£8,500	£9,000
Our market capture	5%	10%	20%
Our scheduler add-on cost	£1,200	£1,500	£1,500
Our revenue	£210,000	£600,000	£1,350,000

Table C.3: Profitability scenario for first three years of business with a marketable product. The revenue is estimated based on the Grid IT sector market capitalisation, number and price of core scheduling units shipped and the market capture percentage and unit price of out scheduler add-on.

release. The salary budgeted for is an industry average, but the company can further benefit from the close links with academic institutions and perhaps gain access to the knowledgeable staff at a reduced cost.

Sales and Admin staff: Until the initial product development cycle is successfully completed, only a very limited sales and admin staff support is needed. A single salesperson can start building up a list of potential clients during this period and engage in marketing the new approach to the scheduling problem. A part-time administration staffer can take care of the salaries, disbursements and basic company paperwork with the help from the management.

Management: A good project manager with the experience in the software development would help the software team stay on track and schedule. Alongside a basic salary, an equity sharing package may be used to attract a committed and worthy candidate.

Fixed Operating Costs: The company will require a substantial investment in the computer hardware and software equipment. This expense can be minimised by using open-source and free software common to the University research community. Office space should be rented, and furniture preferably bought on lease to reduce the amount of cash used. As a new company, suppliers may not be willing to offer lease or credit terms, in which case cooperation with the University can provide basic equipment and furbished offices as part of the seed capital investment or in exchange for an equity in the company.

Cost of Goods Sold: Initially only a small allocation will be made for basic marketing efforts. Software distribution and customer support costs will not be present until the software development phase has been completed.

Revenues: No revenue, except from a possible short contract consultancy work by the management, is anticipated in the first year of operation, or until the release of the first version of the software.

Second and Subsequent Years

Before the launch of the first version of the predictive scheduler add-on, the company can start building its sales force and increase its marketing spending. It is common to release technology preview and beta versions of the new software to demonstrate its functionality to potential clients. In this way, their feedback can be incorporated into the final version, their interest can be judged in advance and estimates can be made on the initial product sales.

Human Costs: The employee structure of the company will likely change with an increases in the sales and administrative staff levels and a reduction in the number of contracted R&D personnel. A Sales and Marketing Manager, and an Operations Manager may also be recruited at this stage to help the entrepreneur regain focus on the technology strategy aspects of the business.

Fixed Operating Costs: The rise of the number of employees will require additional office space and equipment, but with a steady stream of revenues the company should be eligible for trade finance or credit.

Cost of Goods Sold: A substantial part of the gross revenues will go toward customer support and training. Due to the nature of the target hardware and applications, this will require highly skilled staff, able to deal with complex issues of software deployment, interoperability and fault finding on parts of client's critical infrastructure. Software duplication, packaging and distribution expenses will be minimised by offering incentives for buying the software online.

The above can be summarised in the following Table C.6.3 giving the financial outlook for the product development year and the following three years in which the marketable product is bringing in revenues. The analysis indicates that the firm would require around £300,000 to sustain itself until in breaks into profitability. The following section will discuss possible sources from which such funds could be secured.

C.6.4 Sources of Funding

It is clear from the preceding section that a significant investment is needed to support the start-up company before it becomes profitable. This money could come from a number of sources, and would usually be traded for equity in the start-up company. The funds are rarely made available as a lump sum, they are more often paid in instalments and conditional on hitting certain milestones in the product development, product sales or revenues.

This section will not try to give the details of specific funding opportunities, but present an overview of possible funding opportunities and institutions. A

	Year 0	Year 1	Year 2	Year 3
R&D	£175,000	£75,000	£80,000	£120,000
Sales	£30,000	£45,000	£45,000	£60,000
Admin	£20,000	£30,000	£30,000	£45,000
Management	£40,000	£60,000	£60,000	£100,000
Sub-Total HR	£265,000	£210,000	£215,000	£325,000
PC Equipment	£25,000	£10,000	£10,000	£25,000
Offices	£18,000	£20,000	£20,000	£30,000
Furniture	£5,000	£1,000	£2,000	£10,000
Rates	£2,000	£2,000	£2,000	£3,000
Sub-Total Fixed	£50,000	£33,000	£34,000	£68,000
Software distribution	£0	£875	£2,000	£4,500
Customer support	£0	£2,625	£4,000	£13,500
Marketing expenses	£5,000	£10,000	£12,000	£15,000
Sub-Total COGS	£5,000	£13,500	£18,000	£33,000
Total expenses	£320,000	£256,500	£267,000	£426,000
Revenues (Table C.6.2)	£80,000	£210,000	£600,000	£1,350,000
EBITDA	-£240,000	-£46,500	£333,000	£924,000

Table C.4: Four year financial outlook

more detailed survey is deferred until a detailed business plan is available and possible collaborators and partners identified.

Personal or family funds are often used to jump start a company or a product development cycle. They are usually given as a loan with few or no guarantees, sometimes for a share of equity in the new business. The author has a small sum of family savings which he could use to support himself and thus avoid drawing a salary from the company.

University technology transfer programmes give access to funds made available by the University or similar institutions to commercialise research work and create research spin-offs. These programmes can additionally provide office space, equipment and access to skilled labour (students or academics). These funds are relatively modest in size, but the terms are flexible and the author would strongly pursue such funding opportunities.

Bank business or personal loans could be a source of low cost funds not requiring the entrepreneur to give up a share of equity. However, bank's adversity to risk makes these loans hard to get, and often requires a personal guarantee jeopardising owner's personal and family assets. It is unlikely that the author would be granted a bank loan for this particular venture.

Venture capital (VC) is the most frequently used funding source in supporting the technology start-ups. The VCs or individual "angels" can provide

large sums of money and are risk tolerant. They do require a substantial part of the equity in the firm and may impose a management structure to ensure their interest is looked after. Good venture capital is not easy to attract and needs a good business plan and strong marketing. The author would be very receptive to VC funding and would actively seek to attract interest from the individual investors.

C.7 Further Research Proposal

The author has submitted a research proposal to BT Group plc. as part of their Short-term Research Fellowship scheme. The proposal uses the methods and approaches developed in the course of this PhD research to facilitate the management of large Grid clusters and to increase the profitability of commercial Grid service clusters by using a yield management approach.

The research proposal is included as an example of the broader applications of the work presented in this thesis, and as a basis for further research aimed at the commercial use of the predictive, autonomous Grid scheduling.

Improving Service Cluster Profitability Using Yield Management Methods

by Aleksandar Lazarević

Project Summary

Commercial operators of large (Grid) clusters are increasingly offering compute, storage and network resources as a service charged on a per-use basis. From the operator's perspective, maximising the profitability of such an expensive resource usually means striking the right balance between high utilisation levels and acceptable quality of service offered to the consumer. This work proposes a novel way of improving the cluster profitability by analysing the historic workload and inferring the characteristics of specific user behaviour, job arrival rates and execution time patterns. This business intelligence is used to develop a yield management system increasing the overall cluster utilisation by introducing price differentiation. Paired with a pricing policy, the probabilistic workload model increases cluster revenues by making autonomous decisions on job admission and resource reservation in anticipation of the short-term demand behaviour.

Background and Motivation

Sun, HP, Amazon and other leading IT companies are deploying a new business model for computing in which computational and storage resources are made available to the user on an as-needed basis. The goal is to provide a service which would minimise user costs while maximising the efficient use of cluster operator resources. By significantly lowering entry and exit barriers, this utility computing concept is a potentially disruptive technology for present hardware/software vendors and integrators alike.

Profitability of a cap-ex intensive service business greatly depends on the optimal use of its resources. Yield management approach, popularised by the airline industry, is a process of collecting resource usage data, analysing and understanding user behaviour, and reacting to the anticipated demand in order to maximise the profits. The overall goal is to increase revenue by balancing the demand variance through the use of price or service level discrimination.

The proposed research would look at ways of enabling the use of yield management approaches in a utility compute cluster. Central to this effort is an in-depth understanding of the demand presented to the cluster and the ability to effectively forecast its short-term development.

Proposed Methodology

The proposed work is an extension of the author's research into workload characterisation and predictive job scheduling in general purpose utility Grid clusters. The basis of the analysis is the detailed three year workload log from a Grid cluster at the University College London, a European Grid member institution. This rare data from a production Grid, using the same middleware as the Sun Grid Compute Utility – the world's first true compute utility[1], contains more than 3 million jobs from 50+ users in 30 Virtual Organisations compromising academic bodies and their commercial collaborators. The workload is highly heterogeneous, with job execution times ranging from one to 10^7 seconds, and a wide range of workload patterns. Lightly anonymised, it preserves functional dependency between observed metrics and is strongly representative of the demand that a typical utility cluster may experience. The author will make this workload available for the research proposed herein.

[1] Sun Grid Compute Utility - http://www.sun.com/service/sungrid/index.jsp

Objectives

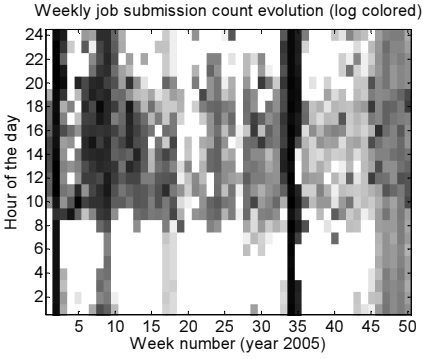
The overall objective of the research is to investigate yield management methods for increasing revenues from a utility compute cluster through selective job admission and price differentiation. More specifically, the following objectives will be pursued:

1. Confirm the presence of cyclic behaviour, temporal patterns and correlations in a representative utility cluster workload
2. Consider different statistical methods for modelling such behaviour in the context of service demand predictions
3. Develop a pricing methodology to support balancing of demand and service price differentiation
4. Develop an admission policy based on the predictive job arrival model to prioritise high-value jobs
5. Validate the proposed approach through simulation using real-world utility cluster workload

Work Programme

The following six week work programme compromising 3 work packages is proposed.

WP 1: Examination of workload behaviour and prediction model selection (1 week) compromises objectives 1-2 and re-examines workload properties and models previously identified by the author in the new context of demand prediction. An example of observed workload behaviour in the figure shows the number of submitted jobs (colour intensity) in each hour of the day over a 51 week period.



WP 2: Yield management and admission policy implementation (3 weeks) is the primary focus of the research in order to accomplish objectives 3-4. Online revenue optimisation will be based on a job control heuristic deciding whether it is more profitable to accept a job being currently offered or block the resources in anticipation of a higher-value job. The approach will be based on a short term load prediction model whose inputs are the current state of the cluster, job meta properties and a historical probability distribution of a certain class of jobs occurring. An offline yield management component will investigate dynamic pricing models that would lead to increased revenues, more balanced demand and higher overall utilisation.

WP 3: Approach validation and result publication (2 weeks) will use the workload log from a representative Grid cluster to test the developed approach using a trace-replay method. Research results will be submitted for publication to a relevant peer-reviewed conference.

Deliverables

1. Summary of job arrival and execution time properties, patterns and correlations of a representative utility cluster workload
2. A pricing methodology and admission policy for maximising service cluster revenue based on a short term demand prediction model
3. Best practices document for cluster monitoring and historical data analysis
4. Research paper submitted to a peer-reviewed conference or journal

C.8 Summary and Conclusions

The analysis of the research work done as part of this PhD thesis showed that significant potential for its commercialisation exists. The opportunity to develop a novel method for scheduling user jobs on large computational Grids unlocks a substantial added value to commercial Grid operators looking to increase their platform utilisation, as well as users looking for a more efficient, convenient and cost effective way of fulfilling their computational needs.

Despite the potential, extracting this added value may prove to be difficult mainly due to the high barriers to entry created by consolidated buyers, high switching costs and brand loyalty. In this environment the most promising commercialisation route would be to develop a predictive scheduling add-on for a third-party Grid scheduler already widely in use. This approach leverages the proprietary know-how obtained during the university research work and minimises the risk associated with the outright competition with an established middleware supplier that would be present if a fully fledged scheduler was developed.

In a dynamic market conditions with many rivals of unequal capitalisation, the best exit strategy for an innovative small company and its founder could be a client or competitor buy-out. The valuation of the business at that point would depend strongly on the level of product development and a commitment by an early client. The management should thus focus on achieving these two as soon as possible.

List of Abbreviations

Abbreviation	Description
AppLeS	Application Level Scheduling
ASCI	Accelerated Strategic Computing Initiative
CCC	UCL Central Computing Cluster
CF	RRD Database Consolidation Function
CPU	Central Processing Unit
DEC	Digital Equipment Corporation (now part of HP)
DS	RRD Database Data Source
FIFO	First In First Out
FLOPS	Floating Point Instructions Per Second
FRFO	First Ready First Out
GASS	Globus Access to Secondary Storage
GGF	Global Grid Forum
GIIS	Grid Information Index Service
GIS	Globus Information Service
GMA	Grid Monitoring Architecture
GRAM	Globus Resource Allocation Manager
GRIS	Grid Resource Information Service
GSI	Globus Security Infrastructure
IP	Internet Protocol
LDAP	Lightweight Directory Access Protocol
LSF	Load Sharing Facility
MDS	Globus Monitoring & Discovery Service
MIPS	Millions of Instructions Per Second
MPI	Message Passing Interface
NWS	Network Weather Service
OGSA	Open Grid Services Architecture
PBS	Portable Batch System
PDF	Probability Distribution Function
PE	GridSim Processing Elements
PID	Process Identifier
PKI	Private Key Infrastructure
RDBMS	Relational Database Management System

... continued on next page

Abbreviation	Description
R-GMA	Relational Grid Monitoring Architecture
RRA	Round Robin Archive
RRD	Round Robin Database
SGE	Sun Grid Engine
SLA	Service Level Agreement
SLAM	SO-GRM SLA Management Component
SMP	Symmetric Multiprocessor
SOAP	Simple Object Access Protocol
SORD	Self-Organised Resource Discovery Protocol
SQL	Simple Query Language
SSH	Secure Shell
Tcl	Tool Command Language
TCP	Transport Control Protocol
TLS	Transport Layer Security
ToS	Type of Service
UDP	User Datagram Protocol
URI	Universal Resource Identifier
VO	Virtual Organisation
WSRF	Web Services Resource Framework
XDR	External Data Representation
XML	eXtensible Mark-up Language

Bibliography

[1] T. Exchange, *Chicago Mercantile Exchange Rulebook*. CME, 2007, vol. Chapter 95.

[2] A. Ingold, I. Yeoman, and U. McMahon, *Yield Management: Strategies for the Service Industries*, 2nd ed. Int. Thomson Business Press, 2001.

[3] J. Subramanian, S. Stidham Jr, and C. Lautenbacher, "Airline yield management with overbooking, cancellations, and no-shows," *Transportation Science*, vol. 33, no. 2, pp. 147–167, 1999.

[4] A. Grimshaw and W. Wulf, "The legion vision of a worldwide virtual computer," *Communications of the ACM*, vol. 40, no. 1, pp. 39–45, 1997.

[5] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@ home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[6] Y. Oyanagi, "Future of supercomputing," *Journal of Computational and Applied Mathematics*, vol. 149, no. 1, pp. 147–153, 2002.

[7] P. Galison and B. Hevly, *Big Science: The Growth of Large-scale Research*. Stanford University Press, 1992.

[8] Unknown, "The politics of grid: Organizational politics as a barrier to implementing grid computing," Platform Computing, Tech. Rep., 2004.

[9] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International Journal of Supercopmuter Applications*, vol. 15(3), 2001.

[10] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," in *Global Grid Forum*, 2002.

[11] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Software Practice and Experience*, vol. 32, no. 2, pp. 135–164, 2002.

[12] R. Wolski, J. Plank, J. Brevik, and T. Bryan, "Analyzing market-based resource allocation strategies for the computational grid," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, p. 258, 2001.

[13] C. Anderson, *The Long Tail: Why the Future of Business Is Selling Less of More.* Hyperion, 2006.

[14] J. Hutchinson, *Fractals and Self Similarity.* University of Melbourne, 1979.

[15] D. Tsafrir and D. Feitelson, "Instability in parallel job scheduling simulation: the role of workload flurries," in *20th International Parallel & Distributed Processing Symposium*, 2006.

[16] Editorial, "10 emerging technologies that will change your world," *Technology Review*, pp. 02–, 2004.

[17] C. Lee, J. Stepanek, R. Wolski, C. Kesselman, and I. Foster, "A network performance tool for grid environments," in *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, 1999.

[18] S. Matsuoka, M. Sato, H. Nakada, and S. Sekiguchi, "Design issues of network enabled server systems for the grid," *Lecture notes in computer science*, no. 1971, pp. 4–17, 2000.

[19] S. Vazhkudai and J. Schopf, "Predicting sporadic grid data transfers," *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings.*, pp. 188–196, 2002.

[20] S. Akioka and Y. Muraoka, "Extended forecast of cpu and network load on computational grid," in *2004 IEEE International Symposium on Cluster Computing and the Grid, 19-22 April 2004.* IEEE, 2004, pp. 765–72.

[21] R. Wolski, N. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *Future Gener. Comput. Syst.*, vol. 15, no. 5-6, pp. 757–768, 1999.

[22] R. Wolski, "Forecasting network performance to support dynamic scheduling using the network weather service," *High Performance Distributed Computing, 1997. Proceedings.*, pp. 316–325, 1997.

[23] W. Cirne and F. Berman, "A comprehensive model of the supercomputer workload," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization, 2 Dec. 2001.* IEEE, 2001, pp. 140–8.

[24] U. Lublin and D. Feitelson, "The workload on parallel supercomputers: modeling the characteristics of rigid jobs," *Journal of Parallel and Distributed Computing T3 - J. Parallel Distrib. Comput. (USA)*, vol. 63, no. 11, pp. 1105–1122, 2003.

[25] A. Downey and D. Feitelson, "The elusive goal of workload characterization," *Performance Evaluation Review T3 - Perform. Eval. Rev. (USA)*, vol. 26, no. 4, pp. 14–29, 1999.

[26] H. Li, D. Groep, and L. Wolters, "Workload characteristics of a multi-cluster supercomputer," in *Job Scheduling Strategies for Parallel Processing. 10th International Workshop, JSSPP 2004. Revised Selected Papers, 13 June 2004.* Springer-Verlag, 2004, pp. 176–93.

[27] E. Medernach, "Workload analysis of a cluster in a grid environment," in *Job Scheduling Strategies for Parallel Processing. 11th International Workshop, JSSPP 2005. Revised Selected Papers, 19 June 2005.* Springer-Verlag, 2005, pp. 36–61.

[28] M. Dobber, R. van der Mei, and G. Koole, "Statistical properties of task running times in a global-scale grid environment," in *Sixth IEEE International Symposium on Cluster Computing and the Grid, 16-19 May 2006.* IEEE Comput. Soc, 2006.

[29] F. Gagliardi, B. Jones, F. Grey, M. Bégin, and M. Heikkurinen, "Building an infrastructure for scientific grid computing: status and goals of the egee project," *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, vol. 363, no. 1833, pp. 1729–1742, 2005.

[30] R. Buyya, D. Abramson, and J. Giddy, "An economy driven resource management architecture for computational power grids," in *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA2000)*, 2000.

[31] R. Buyya, J. Giddy, and D. Abramson, "An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications," in *The Second Workshop on Active Middleware Services (AMS 2000), In conjuction with Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC 2000).* Kluwer Academic Press, 2000.

[32] R. Buyya, M. Murshed, D. Abramson, and S. Venugopal, "Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost-time optimization algorithm," *SoftwarePractice & Experience*, vol. 35, no. 5, pp. 491–512, 2005.

[33] J. Yu, R. Buyya, and C. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," *Proceedings of the 1st International Conference on e-Science and Grid Computing (e-Science 2005)*, pp. 140–147, 2005.

[34] C. Ernemann, V. Hamscher, and R. Yahyapour, "Economic scheduling in grid computing," in *Job Scheduling Strategies for Parallel Processing. 8th International Workshop, JSSPP 2002. Revised Papers, 24 July 2002.* Springer-Verlag, 2002, pp. 128–52.

[35] T. Sandholm, J. Ortiz, J. Odeberg, and K. Lai, "Market-based resource allocation using price prediction in a high performance computing grid for scientific applications," *High Performance Distributed Computing, 15th IEEE International Symposium on*, pp. 132–143, 2006.

[36] L. Sacks, O. Prnjat, I. Liabotis, T. Olukemi, A. Ching, M. Fisher, P. Mckee, N. Georgalas, and H. Yoshii, "Active robust resource management in cluster computing using policies," *Journal of Network and Systems Management*, vol. 11, no. 3, pp. 329–350, 2003.

[37] I. Liabotis, O. Prnjat, T. Olukemi, A. Lazarevic, L. Ching, L. Sacks, M. Fisher, and P. McKee, "Self-organising management of grid resources," in *International Conference on Telecommunications (IST2003)*, 2003.

[38] I. Foster and C. Kesselman, "Globus: a metacomputing infrastructure toolkit," *International Journal of High Performance Computing Applications*, vol. 11, no. 2, p. 115, 1997.

[39] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," *Journal of Computer Science and Technology*, vol. 21, no. 4, pp. 513–520, 2006.

[40] I. Liabotis, O. Prnjat, and L. Sacks, "Policy-based resource management for application level active networks," in *2nd Latin American Network Operations and Management Symposium (LANOMS 2001)*, August 2001.

[41] D. Watts, *Small Worlds*. Princeton University Press, 1999.

[42] T. Olukemi, I. Liabotis, O. Prnjat, and L. Sacks, "Security and resource policy-based mangagement architecture for alan servers," in *Conference on Network Control and Engineering for QoS, Security and Mobility IFIP TC6 (Net-Con 2002)*, 2002, pp. 91–102.

[43] O. Prnjat, T. Olukemi, I. Liabotis, and L. Sacks, "Integrity and security of the application level active networks," in *IFIP Workshop on IP and ATM Traffic Management*, 2001.

[44] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," in *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, vol. 82, 1998.

[45] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, 2001, pp. 181–194.

[46] A. Iamnitchi and I. Foster, "On fully decentralized resource discovery in grid environments," in *International Workshop on Grid Computing*. IEEE, 2001.

[47] C. Schmidt and M. Parashar, "Flexible information discovery in decentralized distributed systems," in *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, 2003, pp. 226–235.

[48] M. Baker and G. Smith, "Gridrm: an extensible resource monitoring system," *Cluster Computing, 2003. Proceedings.*, pp. 207–214, 2003.

[49] A. Cooke, A. Gray, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Cordenonsi, R. Byrom, L. Cornwall, and A. Djaoui, "The relational grid monitoring architecture: Mediating information about the grid," *Journal of Grid Computing*, vol. 2, no. 4, pp. 323–339, 2004.

[50] J. Frey, T. Tannenbaum, I. Foster, and S. Tuecke, "Condor-g: a computation management agent for multi-institutional grids," *High Performance Distributed Computing, 2001. Proceedings.*, pp. 55–63, 2001.

[51] H. El-Rewini, T. Lewis, and H. Ali, *Task scheduling in parallel and distributed systems*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1994.

[52] H. El-Rewini and T. Lewis, *Distributed and parallel computing.* Manning Publications Co. Greenwich, CT, USA, 1998.

[53] B. Shirazi, M. Wang, and G. Pathak, "Analysis and evaluation of heuristic methods for static task scheduling," *Journal of Parallel and Distributed Computing*, vol. 10, no. 3, pp. 222–2232, 1990.

[54] B. Shirazi, K. Kavi, and A. Hurson, *Scheduling and Load Balancing in Parallel and Distributed Systems.* IEEE Computer Society Press Los Alamitos, CA, USA, 1995.

[55] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* WH Freeman & Co. New York, NY, USA, 1979.

[56] G. Ausiello, *Complexity and approximation.* Springer New York, 1999.

[57] S. Cook, "The complexity of theorem-proving procedures," *Proceedings of the third annual ACM symposium on Theory of computing*, pp. 151–158, 1971.

[58] T. Casavant and J. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *Software Engineering, IEEE Transactions on*, vol. 14, no. 2, pp. 141–154, 1988.

[59] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, "An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 3, pp. 236–247, 2003.

[60] M. Litzkow, M. Livny, and M. Mutka, "Condor-a hunter of idle workstations," *Distributed Computing Systems, 1988.*, pp. 104–111, 1988.

[61] J. Cao and F. Zimmermann, "Queue scheduling and advance reservations with cosy," in *Parallel and Distributed Processing Symposium, 2004.Proceedings.18th International*, 2004.

[62] N. Fujimoto and K. Hagihara, "Near-optimal dynamic task scheduling of independent coarse-grained tasks onto a computational grid," in *Parallel Processing, Proceedings. 2003 International Conference on*, 2003, pp. 391–398.

[63] G. Sabin, S. Vishvesh, and P. Sadayappan, "Assessment and enhancement of meta-schedulers for multi-site job sharing," in *High Performance Distributed Computing, 2005.HPDC-14.Proceedings.14th IEEE International Symposium on*, 2005, pp. 144–153.

[64] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert, "Centralized versus distributed schedulers for multiple bag-of-task applications," in *Proceedings. 20th International Parallel and Distributed Processing Symposium, 25-29 April 2006.* IEEE, 2006.

[65] A. Das and D. Grosu, "Combinatorial auction-based protocols for resource allocation in grids," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005.

[66] H. Dail, H. Casanova, and F. Berman, "A decoupled scheduling approach for the grads program development environment," in *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, 2002, pp. 1–14.

[67] A. Aggarwal and R. Kent, "An adaptive generalized scheduler for grid applications," in *Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications*, 2005, pp. 188–194.

[68] M. Wu and X. Sun, "A general self-adaptive task scheduling system for non-dedicated heterogeneous computing," in *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*, 2003, pp. 354–361.

[69] L. Gong, X. Sun, and E. Watson, "Performance modeling and prediction of nondedicated network computing," *IEEE Transactions on Computers*, vol. 51, no. 9, pp. 1041–1055, 2002.

[70] X. Sun and M. Wu, "Grid harvest service: a system for long-term, application-level task scheduling," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003.

[71] A. Su, F. Berman, R. Wolski, and M. Strout, "Using apples to schedule simple sara on the computational grid," *International Journal of High Performance Computing Applications*, vol. 13, no. 3, pp. 253–262, 1999.

[72] J. Subhlok, P. Lieu, and B. Lowekamp, "Automatic node selection for high performance applications on networks," in *Proceedings of the seventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, 1999, pp. 163–172.

[73] K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, and W. Deng, "New grid scheduling and rescheduling methods in the grads project," in *Parallel and Distributed Processing Symposium, 2004.Proceedings.18th International*, 2004.

[74] R. van Nieuwpoort, T. Kielmann, and H. Bal, "Efficient load balancing for wide-area divide-and-conquer applications," in *Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, 2001, pp. 34–43.

[75] A. Abdul-Rahman and S. Hailes, "A distributed trust model," *Proceedings of the workshop on New security paradigms*, pp. 48–60, 1997.

[76] C. Ernemann, V. Hamscher, A. Streit, and R. Yahyapour, "Enhanced algorithms for multisite scheduling," in *Grid Computing - GRID 2002. Third International Workshop. Proceedings, 18 Nov. 2002.* Springer-Verlag, 2002, pp. 219–31.

[77] Y. Zhu, L. Xiao, L. Ni, and Z. Xu, "Incentive-based p2p scheduling in grid computing," in *Proc. of the 3rd International Conference on Grid and Cooperative Computing (GCC2004)*, 2004.

[78] G. Owen, *Game theory.* MIT Press, 1991.

[79] L. Young, S. McGough, S. Newhouse, and J. Darlington, "Scheduling architecture and algorithms within the iceni grid middleware," in *Proc. of the UK e-Science All Hands Meeting*, 2003.

[80] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[81] S. Kirkpatrick, C. Gelatt Jr, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, p. 671, 1983.

[82] M. Aggarwal, R. Kent, and A. Ngom, "Genetic algorithm based scheduler for computational grids," *Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*, pp. 209–215, 2005.

[83] S. Kim and J. Weissman, "A genetic algorithm based approach for scheduling decomposable data grid applications," in *Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04)*, 2004, pp. 406–413.

[84] S. Song, Y. Kwok, and K. Hwang, "Security-driven heuristics and a fast genetic algorithm for trusted grid job scheduling," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, 2005.

[85] N. Spring and R. Wolski, "Application level scheduling of gene sequence comparison on metacomputers," in *Proceedings of the 12th international conference on Supercomputing*, 1998, pp. 141–148.

[86] N. Coleman, R. Raman, M. Livny, and M. Solomon, "Distributed policy management and comprehension with classified advertisements," University of Wisconsin - Madison Computer Sciences Department, Tech. Rep., 2003.

[87] Z. Xuechai, J. Freschl, and J. Schopf, "A performance study of monitoring and information services for distributed systems," *High Performance Distributed Computing, 2003. Proceedings.*, pp. 270–281, 2003.

[88] W. Gentzsch, "Sun grid engine: Towards creating a compute power grid," in *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, 2001.

[89] L. Gong, "Jxta: A network programming environment," *IEEE Internet Computing*, vol. 5, no. 3, pp. 88–95, 2001.

[90] T. Sloan, "Going global with globus and grid engine," *EPCC News*, vol. 48, 2003.

[91] D. A. Lifka, "The anl/ibm sp scheduling system," in *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. London, UK: Springer-Verlag, 1995, pp. 295–303.

[92] J. Skovira, W. Chan, H. Zhou, and D. Lifka, "The easy-loadleveler api project," *Job Scheduling Strategies for Parallel Processing*, pp. 41–47, 1996.

[93] R. Henderson, "Job scheduling under the portable batch system," in *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 1995, pp. 279–294.

[94] S. Zhou, "Lsf: Load sharing in large-scale heterogeneous distributed systems," in *Workshop on Cluster Computing*, 1992.

[95] D. Jackson, Q. Snell, and M. Clement, "Core algorithms of the maui scheduler," *Lecture Notes in Computer Science*, vol. 2221, p. 87, 2001.

[96] E. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-level scheduling on distributed heterogeneous networks," *Supercomputing*, vol. '96, 1996.

[97] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov, "Adaptive computing on the grid using apples," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 14, no. 4, pp. 369–382, 2003.

[98] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: an architecture for a resource management and scheduling system in a global computational grid," in *High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on*, 2000, pp. 283–289.

[99] D. Abramson, R. Sosic, J. Giddy, and B. Hall, "Nimrod: a tool for performing parametrised simulations using distributed workstations," in *High Performance Distributed Computing, 1995., Proceedings of the Fourth IEEE International Symposium on*, 1995, pp. 112–121.

[100] D. Spooner, S. Jarvis, J. Cao, S. Saini, and G. Nudd, "Local grid scheduling techniques using performance prediction," *IEE Proceedings-Computers and Digital Techniques T3 - IEE Proc., Comput. Digit. Tech. (UK)*, vol. 150, no. 2, pp. 87–96, 2003.

[101] S. Jarvis, D. Spooner, H. Keung, J. Dyson, Z. Lei, and G. Nudd, "Performance-based middleware services for grid computing," in *Autonomic Computing Workshop, 2003*, 2003, pp. 151–159.

[102] S. Jarvis, D. Spooner, H. Keung, and G. Nudd, "Performance prediction and its use in parallel and distributed computing systems," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003.

[103] C. Junwei, D. Spooner, S. Jarvis, S. Saini, and G. Nudd, "Agent-based grid load balancing using performance-driven task scheduling," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003.

[104] N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington, "Iceni: An open grid service architecture implemented with jini," in *Supercomputing, ACM/IEEE 2002 Conference*, 2002.

[105] S. Handley, "On the use of a directed acyclic graph to represent a population of computer programs," *The 1 st IEEE Conference on Evolutionary Computation*, pp. 154–159, 1994.

[106] J. Armstrong, *Principles of Forecasting: A Handbook for Researchers and Practitioners*. Kluwer Academic Pub, 2001.

[107] P. Dinda and D. O'Hallaron, "An evaluation of linear models for host load prediction," in *High Performance Distributed Computing, 1999.Proceedings.The Eighth International Symposium on*, 1999, pp. 87–96.

[108] Y. Lingyun, I. Foster, and J. Schopf, "Homeostatic and tendency-based cpu load predictions," *Parallel and Distributed Processing Symposium, 2003. Proceedings.*, p. 9, 2003.

[109] A. Downey, "Predicting queue times on space-sharing parallel computers," in *Parallel Processing Symposium*, 1997, pp. 209–218.

[110] R. Wolski, "Dynamically forecasting network performance using the network weather service," *Cluster Computing*, vol. 1, no. 1, pp. 119–132, 1998.

[111] H. Keung, J. Dyson, S. Jarvis, and G. Nudd, "Predicting the performance of globus monitoring and discovery service (mds-2) queries," in *Grid Computing, 2003. Proceedings. Fourth International Workshop on*, 2003, pp. 176–183.

[112] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Pub, 1997.

[113] J. Peterson and A. Silberschatz, *Operating system concepts*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1985.

[114] A. Burns and A. Wellings, *Real-time systems and their programming languages*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1990.

[115] G. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer, 2005.

[116] C. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are user runtime estimates inherently inaccurate?" *Job Scheduling Strategies for Parallel Processing*, 2004.

[117] F. Harary, *Graph Theory*. Perseus Books, 1999.

[118] V. Sarkar, "Determining average program execution times and their variance," in *SIGPLAN Conference on Programming Language Design and Implementation*, 1989, pp. 298–312.

[119] R. Huang, H. Casanova, and A. Chien, "Using virtual grids to simplify application scheduling," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006.

[120] M. Gergeleit, E. Nett, and J. Fitzner, "On-line prediction of execution times - a basis for adaptive scheduling," in *Object-Oriented Real-Time Dependable Systems, 1999.Proceedings.Fourth International Workshop on*, 1999, pp. 186–194.

[121] D. Kerbyson, "Predictive performance and scalability modeling of a large-scale application," in *Conference on High Performance Networking and Computing*. ACM Press, 2001.

[122] G. Marin and J. Mellor-Crummey, "Cross-architecture performance predictions for scientific applications using parameterized models," in *SIGMETRICS 2004*. ACM Press, 2004, pp. 2–13.

[123] J. Henning, "Spec cpu2000: Measuring cpu performance in the new millennium," *Computer*, vol. 2000, no. July, 2000.

[124] J. Farmer and J. Sidorowich, "Predicting chaotic time series," *Physical Review Letters*, vol. 59, no. 8, pp. 845–848, 1987.

[125] D. Montgomery and G. Runger, *Applied statistics and probability for engineers*. John Wiley & Sons New York, 1994.

[126] W. Smith, I. Foster, and V. Taylor, "Predicting application run times using historical information," *Lecture Notes in Computer Science*, vol. 1459, p. 122, 1998.

[127] N. Draper and H. Smith, *Applied Regression Analysis*. John Wiley & Sons New York, 1981.

[128] P. Dinda, "Online prediction of the running time of tasks," in *High Performance Distributed Computing, 2001.Proceedings.10th IEEE International Symposium on*, 2001, pp. 383–394.

[129] L. Byoung Dai and J. Schopf, "Run-time prediction of parallel applications on shared environments," *Cluster Computing, 2003. Proceedings.*, pp. 487–491, 2003.

[130] G. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[131] P. Dinda, "A prediction-based real-time scheduling advisor," in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, 2002, pp. 10–17.

[132] J. Doob, *Stochastic processes*. Wiley New York, 1990.

[133] J. Schopf and F. Berman, "Performance prediction in production environments," in *Parallel Processing Symposium, 1998.1998 IPPS/SPDP.Proceedings of the First Merged International...and Symposium on Parallel and Distributed Processing 1998*, 1998, pp. 647–653.

[134] J. Schopf and F. Berman, "Stochastic scheduling," in *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, 1999.

[135] J. Schopf and F. Berman, "Using stochastic intervals to predict application behavior on contended resources," in *Parallel Architectures, Algorithms, and Networks, 1999. (I-SPAN '99) Proceedings. Fourth InternationalSymposium on*, 1999, pp. 344–349.

[136] G. Haring, *On Stochastic Models of Interactive Workloads*. North-Holland, 1983.

[137] G. Serazzi, "A functional and resource-oriented procedure for workload modeling," *Proceedings of the 8th International Symposium on Computer Performance Modeling, Measurement and Evaluation*, p. 345, 1981.

[138] J. Kearns and S. DeFazio, "Diversity in database reference behavior," *Proceedings of the 1989 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 11–19, 1989.

[139] P. Lewis and G. Shedler, "Statistical analysis of non-stationary series of events in a data base system," *IBM Journal of Research and Development*, vol. 20, no. 5, p. 465, 1976.

[140] H. Artis, "Capacity planning for mvs computer systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 8, no. 4, pp. 45–62, 1979.

[141] R. Gusella, "A measurement study of diskless workstation traffic on an ethernet," *Communications, IEEE Transactions on*, vol. 38, no. 9, pp. 1557–1568, 1990.

[142] R. Bodnarchuk and R. Bunt, "A synthetic workload model for a distributed system file server," *ACM SIGMETRICS Performance Evaluation Review*, vol. 19, no. 1, pp. 50–59, 1991.

[143] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of www client-based traces," Boston University, Tech. Rep., 1995.

[144] M. Arlitt and C. Williamson, "Web server workload characterization: the search for invariants," *ACM SIGMETRICS Performance Evaluation Review*, vol. 24, no. 1, pp. 126–137, 1996.

[145] M. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," *IEEE/ACM Transactions on Networking (TON)*, vol. 5, no. 6, pp. 835–846, 1997.

[146] D. Feitelson, "The forgotten factor: Facts on performance evaluation and its dependence on workloads," *Euro-Par*, p. 49, 2002.

[147] D. Feitelson, "Workload modeling for performance evaluation," *Lecture Notes in Computer Science*, pp. 114–141, 2002.

[148] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking T3 - IEEE/ACM Trans. Netw. (USA)*, vol. 2, no. 1, pp. 1–15, 1994.

[149] V. Paxson and S. Floyd, "Wide area traffic: the failure of poisson modeling," *Networking, IEEE/ACM Transactions on*, vol. 3, no. 3, pp. 226–244, 1995.

[150] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pp. 151–160, 1998.

[151] P. Barford and M. Crovella, "Measuring web performance in the wide area," *ACM SIGMETRICS Performance Evaluation Review*, vol. 27, no. 2, pp. 37–48, 1999.

[152] R. Gibbons, "A historical application profiler for use by parallel schedulers," in *Proceedings of 3rd Workshop on Job Scheduling Strategies for Parallel Processing (Held in Conj. with IPPS'97), 5 April 1997.* Springer-Verlag, 1997, pp. 58–77.

[153] M. Calzarossa and G. Serazzi, "Construction and use of multiclass workload models," *Performance Evaluation T3 - Perform. Eval. (Netherlands)*, vol. 19, no. 4, pp. 341–352, 1994.

[154] D. Ferrari, "On the foundations of artificial workload design," *Proceedings of the 1984 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pp. 8–14, 1984.

[155] B. Song, C. Ernemann, and R. Yahyapour, "Parallel computer workload modeling with markov chains," in *Job Scheduling Strategies for Parallel Processing. 10th International Workshop, JSSPP 2004. Revised Selected Papers, 13 June 2004.* Springer-Verlag, 2004, pp. 47–62.

[156] N. Thomas, "Modelling job allocation where service duration is unknown," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006.

[157] A. Iosup, C. Dumitrescu, D. Epema, H. Li, and L. Wolters, "How are real grids used? the analysis of four grid traces and its implications," in *The 7th IEEE/ACM International Conference on Grid Computing (Grid2006).* IEEE Computer Society Press, 2006.

[158] D. Tsafrir, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE transactions on parallel and distributed systems*, vol. 18, no. 6, pp. 789–803, 2007.

[159] K. Li, "Job scheduling for grid computing on metacomputers," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005.

[160] F. Sacerdoti, M. Katz, M. Massie, and D. Culler, "Wide area cluster monitoring with ganglia," in *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*, 2003, pp. 289–298.

[161] B. Tierney, "A grid monitoring architecture," Global Grid Forum, Tech. Rep., 2000.

[162] M. Leese and R. Tasker, "Gridmon and network performance monitoring for the grid," Networkshop, Tech. Rep., 2004.

[163] H. Casanova, "Simgrid: a toolkit for the simulation of application scheduling," *Cluster Computing and the Grid, 2001. Proceedings.*, pp. 430–437, 2001.

[164] A. Legrand, L. Marchal, and H. Casanova, "Scheduling distributed applications: the simgrid simulation framework," *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003.*, pp. 138–145, 2003.

[165] H. Casanova, "Modeling large-scale platforms for the analysis and the simulation of scheduling strategies," in *Parallel and Distributed Processing Symposium, 2004.Proceedings.18th International*, 2004.

[166] R. Buyya and M. Murshed, *GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing*. John Wiley & Sons, Ltd., 2003.

[167] H. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien, "The microgrid: a scientific tool for modeling computational grids," in *Supercomputing*, 2000.

[168] X. Huaxia, H. Dail, H. Casanova, and A. Chien, "The microgrid: using online simulation to predict application performance in diverse grid network environments," *Challenges of Large Applications in Distributed Environments, 2004. CLADE 2004.*, pp. 52–61, 2004.

[169] J. Tukey, *Exploratory data analysis*. Addison-Wesley Menlo Park, CA, 1977.

[170] N. Technology, *NIST/SEMATECH e-Handbook of Statistical Methods*. NIST/SEMANTECH, 2003.

[171] F. Mosteller and J. Tukey, *Data Analysis and Regression: A Second Course in Statistics*. Addison-Wesley Reading, MA, 1977.

[172] P. Velleman and D. Hoaglin, *Applications, Basics, and Computing of Exploratory Data Analysis*. Cornell Cooperative Extension, 1981.

[173] M. Calzarossa and G. Serazzi, "Workload characterization: a survey," *Proceedings of the IEEE T3 - Proc. IEEE (USA)*, vol. 81, no. 8, pp. 1136–1150, 1993.

[174] B. Mandelbrot, *The fractal geometry of nature*. WH Freeman and Co., 1983.

[175] R. Clegg, "A practical guide to measuring the hurst parameter," in *Proc. of 21st UK Performance Engineering Workshop, School of Computing Science Technical Report Series, CS-TR-916, University of Newcastle*, 2005, pp. 1368–2428.

[176] J. Beran, *Statistics for Long-Memory Processes*. CRC Press, 1994.

[177] R. Kriesten, U. Kaage, and F. Jondral, "A unifying view to fractional modeling," in *Global Telecommunications Conference, GLOBECOM'99*, 1999.

[178] A. Chamoli, A. Bansal, and V. Dimri, "Wavelet and rescaled range approach for the hurst coefficient for short and long time series," *Computers & Geosciences*, vol. 33, no. 1, pp. 83–93, 2007.

[179] J. Feder, *Fractals*. Plenum Press, 1989.

[180] C. Turvey, "A note on scaled variance ratio estimation of the hurst exponent with application to agricultural commodity prices," *Physica A: Statistical Mechanics and its Applications*, vol. 377, no. 1, pp. 155–165, 2007.

[181] S. Stoev, M. Taqqu, C. Park, and J. Marron, "On the wavelet spectrum diagnostic for hurst parameter estimation in the analysis of internet traffic," *Computer Networks*, vol. 48, no. 3, pp. 423–445, 2005.

[182] N. Cackov, Z. Lucic, M. Bogdanov, and L. Trajkovic, "Wavelet-based estimation of long-range dependence in mpeg video traces," *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 2068–2071, 2005.

[183] H. Hurst, R. Black, and Y. Simaika, *Long-term Storage: An Experimental Study*. Constable, 1965.

[184] J. Kenney, *Mathematics of Statistics*. Van Nostrand, 1954.

[185] C. Kenyon and G. Cheliotis, "Creating services with hard guarantees from cycle-harvesting systems," *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003.*, pp. 224–231, 2003.

[186] D. Feitelson, "Memory usage in the lanl cm-5 workload," *Job Scheduling Strategies for Parallel Processing*, pp. 78–94, 1997.

[187] M. Calzarossa, G. Haring, G. Kotsis, A. Merlo, and D. Tessera, "A hierarchical approach to workload characterization for parallel systems," in *Proceedings of International Conference on High-Performance Computing and Networking. HPCN '95, 3-5 May 1995*. Springer-Verlag, 1995, pp. 102–9.

[188] D. Pollock, *Handbook of Time Series Analysis, Signal Processing, and Dynamics*. Academic Press, 1999.

[189] L. Ljung, *System identification: theory for the user*. Prentice-Hall, Inc., 1986.

[190] E. Hannan and B. Quinn, "The determination of the order of an autoregression," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 41, no. 2, pp. 190–195, 1979.

[191] E. Parzen, "Some recent advances in time series modeling," *Automatic Control, IEEE Transactions on*, vol. 19, no. 6, pp. 723–730, 1974.

[192] R. Cook and S. Weisberg, *Residuals and influence in regression*. Chapman and Hall New York, 1982.

[193] R. Carbone and J. Armstrong, "Evaluation of extrapolative forecasting methods: Results of a survey of academicians and practitioners," *Journal of Forecasting*, vol. 1, p. 215, 1982.

[194] J. Armstrong, F. Collopy, M. Dept, and W. School, *Error Measures for Generalizing about Forecasting Methods: Empirical Comparisons*. Wharton School, University of Pennsylvania, Marketing Dept, 1990.

[195] J. Armstrong, "Evaluating forecasting methods," *Principles of Forecasting. Norwell, MA, Kluwer Academic Publishers*, pp. 365–382, 2001.

[196] S. Makridakis, *Accuracy measures: theoretical and practical concerns*. INSEAD, 1993.

[197] S. Makridakis, C. Chatfield, M. Hibon, M. Lawrence, T. Mills, K. Ord, and L. Simmons, "The m2-competition: A real-time judgmentally based forecasting study," *International Journal of Forecasting*, vol. 9, no. 1, pp. 5–22, 1993.

[198] D. Swanson, J. Tayman, and C. Barr, "A note on the measurement of accuracy for subnational demographic estimates," *Demography*, vol. 37, no. 2, pp. 193–201, 2000.

[199] R. Hyndman and C. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22(4), pp. 679–688, 2005.

[200] J. Bradley, "Matlab statistics toolbox: Users guide," *The MathWorks Inc*, pp. 77–87, 1997.

[201] D. Talby and D. Feitelson, "Improving and stabilizing parallel computer performance using adaptive backfilling," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005.

[202] J. Stankovic, *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Pub, 1998.

[203] J. Jackson, *Scheduling a Production Line to Minimize Maximum Tardiness*. University of California, 1955.

[204] R. Graham, E. Lawler, J. Lenstra, and A. Rinnooy Kan, "Optimisation and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, no. 236-287, p. 18, 1979.

[205] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[206] A. Mu'alem and D. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp 2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, 2001.

[207] D. Tsafrir, Y. Etsion, and D. Feitelson, "Modeling user runtime estimates," in *Proceedings of 11th Job Scheduling Strategies for Parallel Processing*, 2005.

[208] D. Feitelson, "Experimental analysis of the root causes of performance evaluation results: a backfilling case study," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 175–182, 2005.

[209] S. Chiang, A. Arpaci-Dusseau, and M. Vernon, "The impact of more accurate requested runtimes on production job scheduling performance," *Job Scheduling Strategies for Parallel Processing*, p. 103, 2002.

[210] D. Zotkin and P. Keleher, "Job-length estimation and performance in backfilling schedulers," *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, pp. 236–243, 1999.

[211] D. Tsafrir, "Estimates generator," Computer Software, 2005.

[212] J. Beiriger, H. Bivens, S. Humphreys, W. Johnson, and R. Rhea, "Constructing the asci computational grid," in *Ninth IEEE International Symposium on High Performance Distributed Computing*, 2000.

[213] M. Lamanna, "The lhc computing grid project at cern," *Nuclear Inst. and Methods in Physics Research, A*, vol. 534, no. 1-2, pp. 1–6, 2004.

[214] R. Gardner, "The grid2003 project. the grid3 production grid: Principles and practice," in *Proceedings of the 13th IEEE International Symposium on High-Performance Distributed Computing (HPDC 2004)*, 2004.

[215] C. Catlett, "Teragrid: A foundation for us cyberinfrastructure," *NPC*, 2005.

[216] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[217] C. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1982.

[218] H. Li, D. Groep, J. Templon, and L. Wolters, "Predicting job start times on clusters," in *IEEE International Symposium on Cluster Computing and the Grid*. IEEE, 2004, pp. 301–.

[219] B. Segal, "Grid computing: The european data grid project," *IEEE Nuclear Science Symposium and Medical Imaging Conference*, p. 15, 2000.

[220] H. Li, J. Chen, Y. Tao, D. Groep, and L. Wolters, "Improving a local learning technique for queue wait time predictions," in *Sixth IEEE International Symposium on Cluster Computing and the Grid, 16-19 May 2006*. IEEE Comput. Soc, 2006.

[221] D. Aha, D. Kibler, and M. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991.

[222] D. Talby, D. Tsafrir, Z. Goldberg, and D. Feitelson, "Session-based, estimation-less, and information-less runtime prediction algorithms for parallel and grid job scheduling," School of Computer Science and Engineering, the Hebrew University, Tech. Rep., 2006.

[223] J. Zilber, O. Amit, and D. Talby, "What is worth learning from parallel workloads?: a user and session based analysis," in *Proceedings of the 19th annual international conference on Supercomputing*, 2005, pp. 377–386.

[224] P. Taylor, "The san diego supercomputer center," *Computational Science and Engineering, IEEE*, vol. 1, no. 3, 1995.

[225] S. Hotovy, D. Schneider, and T. O'Donnell, "Analysis of the early workload on the cornell theory center ibm sp2," *Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 272–273, 1996.

[226] A. Takefusa, H. Casanova, S. Matsuoka, and F. Berman, "A study of deadline scheduling for client-server systems on the computational grid," in *Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing*, 2001.

[227] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima, "Overview of a performance evaluation system for global computing scheduling algorithms," *High Performance Distributed Computing, 1999. Proceedings.*, pp. 97–104, 1999.

[228] E. Caron, P. Chouhan, and F. Desprez, "Deadline scheduling with priority for client-server systems on the grid," in *Proceedings. Fifth IEEE/ACM International Workshop on Grid Computing, 8 Nov. 2004.* IEEE Comput. Soc, 2004, pp. 410–14.

[229] D. Abramson, J. Giddy, and L. Kotler, "High performance parametric modeling with nimrod/g: Killer application for the global grid," *International Parallel and Distributed Processing Symposium (IPDPS)*, p. 520, 2000.

[230] M. Gergeleit, *Automatic Instrumentation of Object oriented programs.* Gesellschaft fur Mathematik und Datenverarbeitung, 1994.

[231] K. Templer and C. Jeffery, "A configurable automatic instrumentation tool for ansi c," *Automated Software Engineering, Proceedings. 13th IEEE International Conference on*, pp. 249–258, 1998.

[232] S. Ling and W. Li, "On fractionally integrated autoregressive moving-average time series models with conditional heteroscedasticity," *Journal of the American Statistical Association*, vol. 92, no. 439, 1997.

[233] J. Cao, D. Spooner, S. Jarvis, and G. Nudd, "Grid load balancing using intelligent agents," *Future Generation Computer Systems T3 - Future Gener. Comput. Syst. (Netherlands)*, vol. 21, no. 1, pp. 135–149, 2005.

[234] C. Bishop, *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[235] D. Citron, "Misspeculation: partial and misleading use of spec cpu2000 in computer architecture conferences," *Computer Architecture, 2003. Proceedings.*, pp. 52–59, 2003.

[236] N. Amato and L. Dale, "Probabilistic roadmap methods are embarrassingly parallel," *Robotics and Automation, 1999. Proceedings.*, vol. 1, pp. 688–694, 1999.

[237] S. Andreozzi, "Glue schema implementation for the ldap data model," Technical Report INFN/TC-04/16, INFN, Tech. Rep., 2004.

[238] D. Rusling, "The linux kernel," *Linux Documentation Project*, vol. 1999, 1999.

[239] K. Sovani, "Kernel korner: sleeping in the kernel," *Linux Journal*, pp. 137–, 2005.

[240] M. Porter, *Competitive advantage.* Free Press New York, 1985.